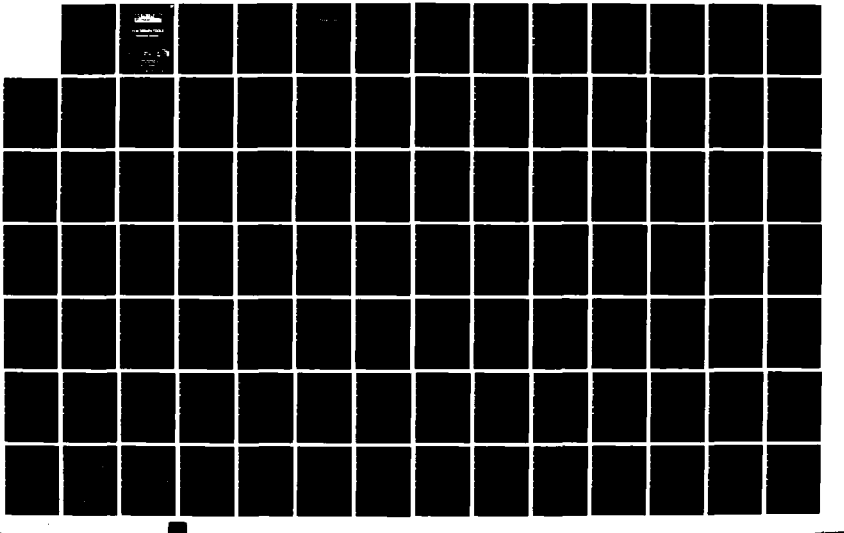
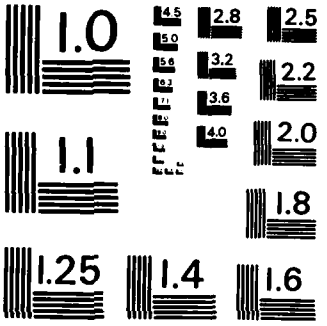
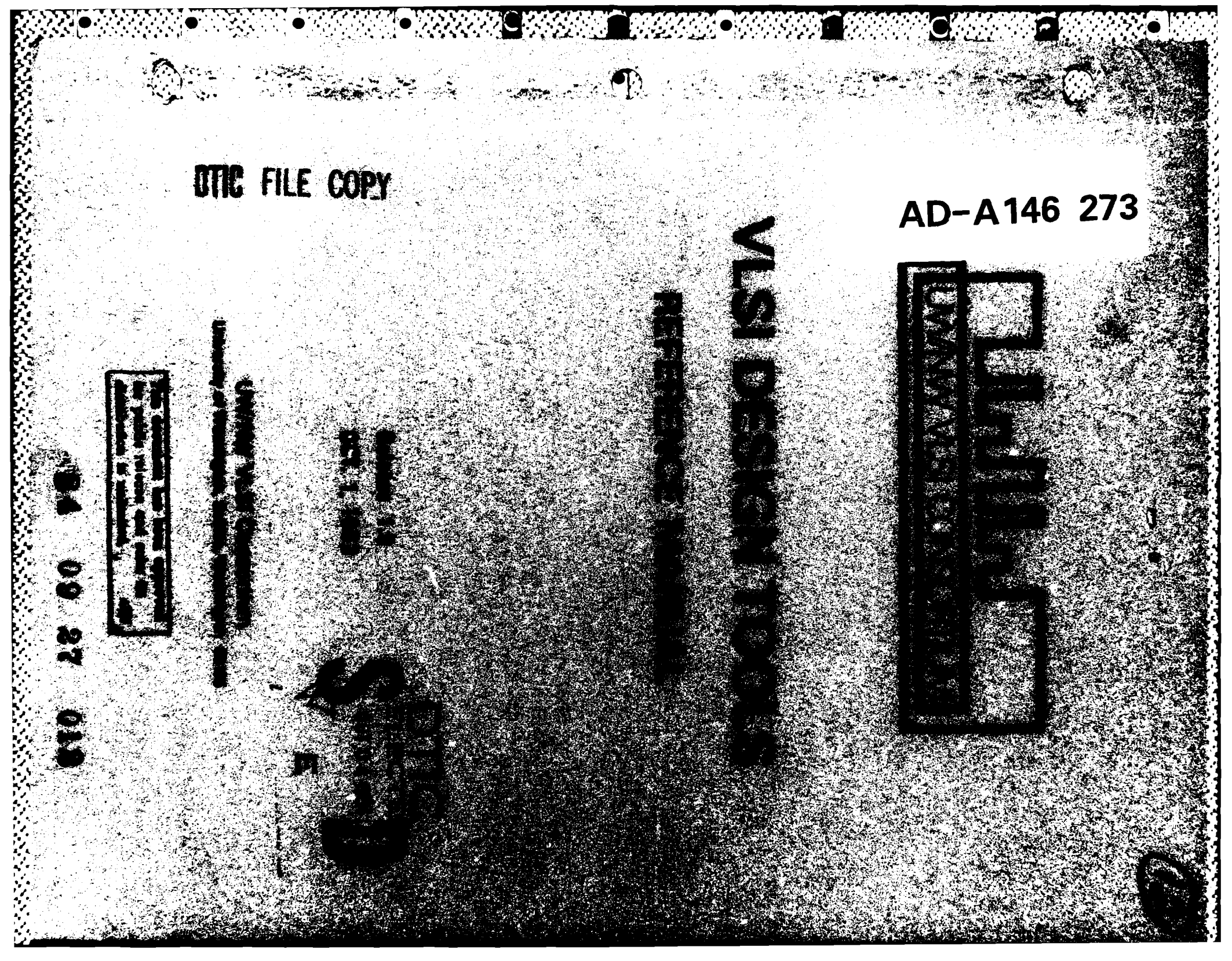
AD-A146 273 VLSI (VERY LARGE SCALE INTEGRATION) DESIGN TOOLS 1/2
REFERENCE MANUAL - RELEASE 10(U) WASHINGTON UNIV
SEATTLE DEPT OF COMPUTER SCIENCE 01 OCT 83 TR-84-08-06
UNCLASSIFIED MDA903-82-C-0424 F/G 9/2 NL







unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR - 84 - 08 - 06	2. GOVT ACCESSION NO. AD-A146 273	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) VLSI DESIGN TOOLS REFERENCE MANUAL - RELEASE 1.0		5. TYPE OF REPORT & PERIOD COVERED Technical, interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) UW/NW VLSI CONSORTIUM		8. CONTRACT OR GRANT NUMBER(s) MDA 903-82-C-0424
9. PERFORMING ORGANIZATION NAME AND ADDRESS UW/NW VLSI Consortium Department of Computer Science, FR-35 University of Washington Seattle, Washington 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N/A
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA - IPTO 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE October 1983 13. NUMBER OF PAGES 151
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ONR University of Washington 315 University District Building 1107 N.E. 45th St., JD-16 Seattle, Washington 98195		15. SECURITY CLASS. (of this report) unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

very large scale integration, VLSI design tools, CAD tools, nMOS, VLSI layout, VLSI circuit simulation, design rule checking

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the use of the University of Washington/Northwest VLSI Consortium's package of VLSI design tools. The tools described are:

(see reverse)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.

Appendix A. Tools for Distribution

1. High Level Circuit Design

PEG	PLA Equation Generator	Gordon Hamachi (UCB)
EQNTOTT	Boolean Equations to Truth Table	Bob Cmelik (UCB)
TPLA	Technology Independent PLA Generator	Robert N. Mayo (UCB)

2. Layout

PLAP	Pascal Layout Artwork Program	Bruce Yanagida (Boeing)
CAESAR	Graphics Layout Editor	John Ousterhout (UCB)
QUILT	Layout Tile Assembly	Robert N. Mayo (UCB)

3. Display

CIFPLOT	CIF Interpreter and raster plotter	Dan Fitzpatrick (UCB)
DBPLOT	Pen Plot of layout	J. E. Thiele (Boeing)
DBDI	Graphics Display of Layout	K. P. Naylor (Boeing)

4. Rule Checking

DRC	Layout Rule Checker	Dorothea Haken (CMU)
LYRA	Layout Rule Checker	Michael Arnold (UCB)
ERC	Electrical Rules Checker	Keith Pennick (Boeing)

5. Simulation

MEXTRA	Manhattan Circuit Extractor	Dan Fitzpatrick (UCB)
ESIM	Logic-level simulator	Chris Terman (MIT/UCB)
SPICE	Electrical Circuit Analysis	Lawrence Nagel (UCB)
SLANG	Functional-level simulator	John Foderaro (UCB)
NL	Logic-level simulator	Chris Terman (MIT)
RNL	Timing simulator	Chris Terman (MIT)
CRYSTAL	Timing analyzer	John Ousterhout (UCB)

Only the major programs are shown above. Also included are other pieces of software that facilitate communication between these programs.

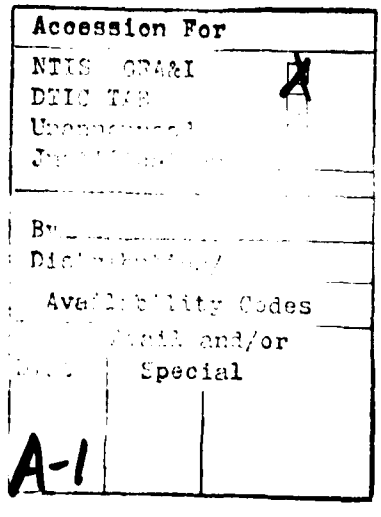
VLSI Design Tools

Reference Manual

Release 1.0

10/1/83

TR-84-08-06



UW/NW VLSI Consortium
Department of Computer Science
Room 315 Sieg Hall
University of Washington FR-35
Seattle, Washington 98195

UW/NW VLSI CONSORTIUM

ACADEMIC DISTRIBUTION AGREEMENT

1. The UW/NW VLSI CONSORTIUM provides the VLSI Software System Package, described in Appendix A, to the academic institution named below (hereinafter referred to as RECIPIENT), under the terms and conditions hereinafter set out.

2. RECIPIENT agrees to furnish distribution medium (magnetic tape).

3. This Software System Package has been tested and is in use at the UW/NW VLSI CONSORTIUM, however, NO WARRANTY, EXPRESSED OR IMPLIED, is furnished with this Software System Package.

4. Problems may be reported to, and assistance received by contacting:

VIA ARPANET: bugs@uw-vlsi

U.S. MAIL: UW/NW VLSI CONSORTIUM
Attn: Merry Bush
Computer Science Department FR-35
University of Washington
Seattle, WA. 98195

TELEPHONE: (206) 545-3796

5. The RECIPIENT agrees that this Software System Package is provided solely for the internal noncommercial use of the RECIPIENT and shall not be distributed or transferred to any person other than the RECIPIENT, nor shall it be exported to any component of the RECIPIENT outside the United States, nor shall the RECIPIENT sell access to the Package.

6. Title and copyright to the Programs in the Package shall at all times remain with the persons and institutions named in Appendix A (description of Software System Package).

7. Nothing in this Agreement shall be construed as conferring rights to use in advertising, publicity or otherwise any trademark or the names of either the UW/NW VLSI CONSORTIUM or the UNIVERSITY OF WASHINGTON. Such use requires prior approval in writing.

RECIPIENT:_____ UW/NW VLSI CONSORTIUM

By_____ By_____

Title_____ Title_____

Date_____ Date_____

NO PURCHASE ORDERS WILL BE ACCEPTED.

Appendix A. Tools for Distribution

1. High Level Circuit Design

PEG	PLA Equation Generator	Gordon Hamachi (UCB)
EQNTOTT	Boolean Equations to Truth Table	Bob Cmelik (UCB)
TPLA	Technology Independent PLA Generator	Robert N. Mayo (UCB)

2. Layout

PLAP	Pascal Layout Artwork Program	Bruce Yanagida (Boeing)
CAESAR	Graphics Layout Editor	John Ousterhout (UCB)
QUILT	Layout Tile Assembly	Robert N. Mayo (UCB)

3. Display

CIFPLOT	CIF Interpreter and raster plotter	Dan Fitzpatrick (UCB)
DBPLOT	Pen Plot of layout	J. E. Thiele (Boeing)
DBDI	Graphics Display of Layout	K. P. Naylor (Boeing)

4. Rule Checking

DRC	Layout Rule Checker	Dorothea Haken (CMU)
LYRA	Layout Rule Checker	Michael Arnold (UCB)
ERC	Electrical Rules Checker	Keith Pennick (Boeing)

5. Simulation

MEXTRA	Manhattan Circuit Extractor	Dan Fitzpatrick (UCB)
ESIM	Logic-level simulator	Chris Terman (MIT/UCB)
SPICE	Electrical Circuit Analysis	Lawrence Nagel (UCB)
SLANG	Functional-level simulator	John Foderaro (UCB)
NL	Logic-level simulator	Chris Terman (MIT)
RNL	Timing simulator	Chris Terman (MIT)
CRYSTAL	Timing analyzer	John Ousterhout (UCB)

Only the major programs are shown above. Also included are other pieces of software that facilitate communication between these programs.

TABLE OF CONTENTS

INTRODUCTION

Who We Are : The UW/NW VLSI Consortium

UW/NW VLSI Consortium members include the University of Washington, represented by the Computer Science Department, and five Pacific Northwest firms: Boeing Aerospace, John Fluke Manufacturing, Tektronix, Honeywell Marine Systems, and Microtel Pacific Research of Canada. The purpose of the Consortium is to advance the state of the art in VLSI technology and to transfer this technology to American industry.

Each corporate member of the Consortium has a full-time liaison on campus who cooperates with faculty members on circuit design as well as filling the role of visiting faculty, working with graduate students and contributing valuable "real world" experience to the Department of Computer Science. This program is serving as a demonstration of cooperative research and technology exchange among universities and industry.

The Consortium maintains a VLSI design system and plans to evolve its capability over time, as well as to provide training in its use. The Consortium validates the system by exercising it with industry design problems. Missing pieces will be identified for guiding future research and development. The resulting system is made available to other universities and industry.

Students from industry and universities are being trained in the VLSI design methodology through a series of regularly scheduled intensive courses. Educational materials are being developed to allow similar training capability to be distributed to other locations throughout the country. Refresher courses, symposia and reports are being scheduled as new capabilities are added.

Installation Procedure

The distribution tape contains VLSI design tools that run on a VAX with Berkeley 4.1 UNIX. Many of them will not run on other machines or other versions of UNIX. In addition, VLSI display tools require plotting or graphics devices:

Pen plotters (HP7221, HP7580)

Dot matrix printers (Versatec, Varian, Printronix)

Interactive graphics devices with bitpads (AED512, Metheus Omega 440)

The tape is in tar format, 1600 bpi. To read the tape, do the following: Create a directory in which the tools are to reside, "chdir" to this directory, mount the tape, and type "tar x". The following subdirectories will be created:.

bin	executables and shell scripts
doc	user manuals
include	source header files for some tools
lib	archived libraries and other stuff
man	UNIX programmer's manual entries for each tool
src	source code, make files, and installation scripts

These will be referred to as tape/bin, tape/doc, etc. in the sequel.

Once the tape is read in, the tools may be installed. The installation procedure (shell script tape/src/INSTALL) moves files from tape/include, tape/lib, and tape/man to /usr/include, /usr/lib, and /usr/man. The installation script also installs a new "man" command. (Man has been changed to support entries with extensions of "1i" in /usr/man/man1 so the VLSI man entries can be easily identified; it also will automatically call cadman if you have the Berkeley VLSI tools installed - see the makefile in src/man.) You may wish to move the executables to another directory rather than keeping them in tape/bin (on our machine they reside in /usr/vlsibin); change the setting of B in the INSTALL script to do this. In either case you will want to make sure the directory containing the executables is in everyone's search path.

To install the tools, "chdir" to tape/src, make sure you are super-user, and type "INSTALL". Read tape/README for further information about installation.

If you have the Winter '83 distribution of the Berkeley VLSI tools, there are several improvements and bug fixes available in tape/src/ucb-cad. These may be installed using the INSTALL-UCB script in tape/src. You may wish to review the README file in tape/src/ucb-cad and edit INSTALL-UCB appropriately if you want only some of the changes. Since only the *changed* source files are included in src/ucb-cad, you must follow the procedures documented with the Berkeley VLSI tools to complete the installation after using INSTALL-UCB.

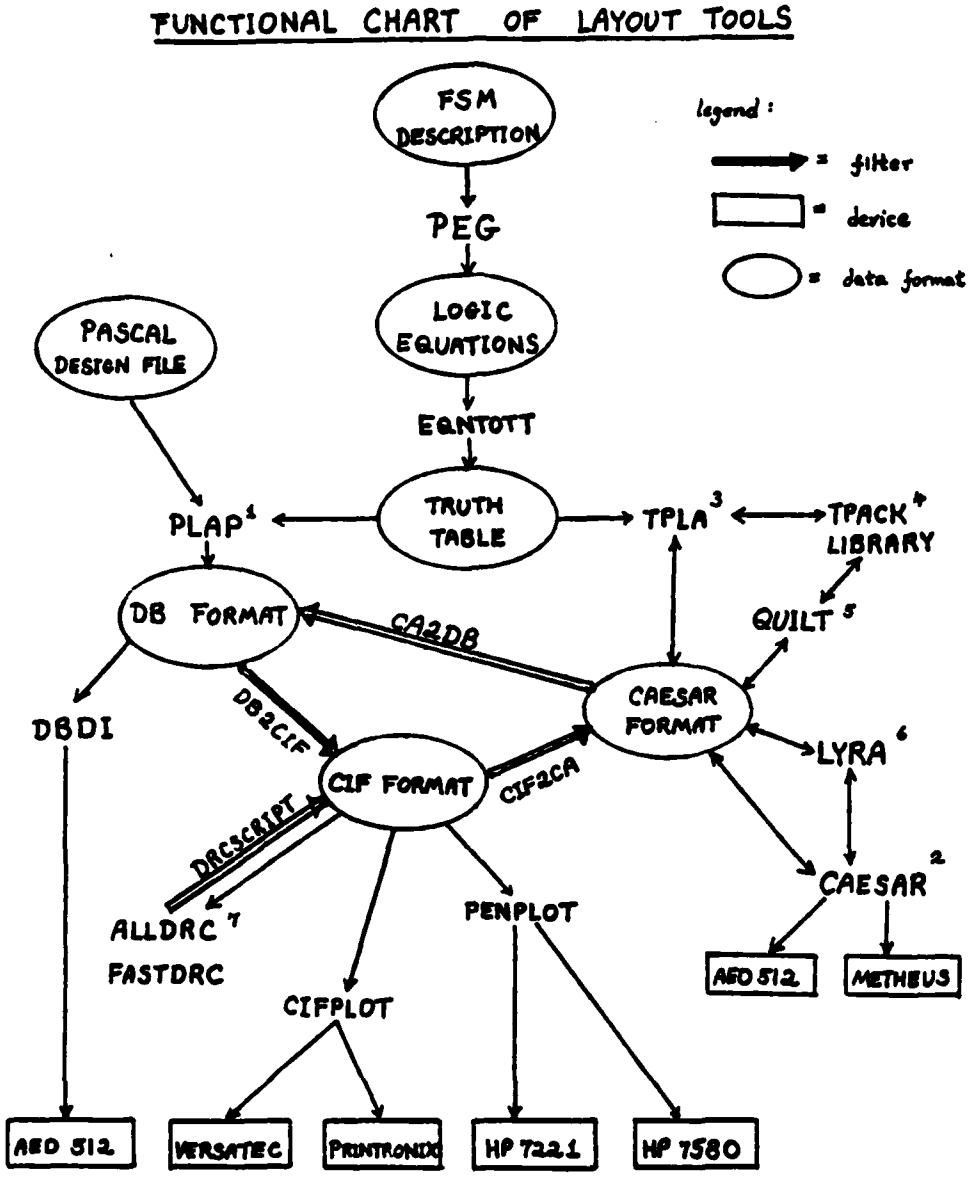
The following device-related sources are included:

Sources for lpr that complement the changes to cifplot and allow cifplot to be used with a Printronix.

Sources for a parallel interface device driver for a Metheus omega 440 color graphics display. A DEC DR-11W is needed in addition to the Metheus.

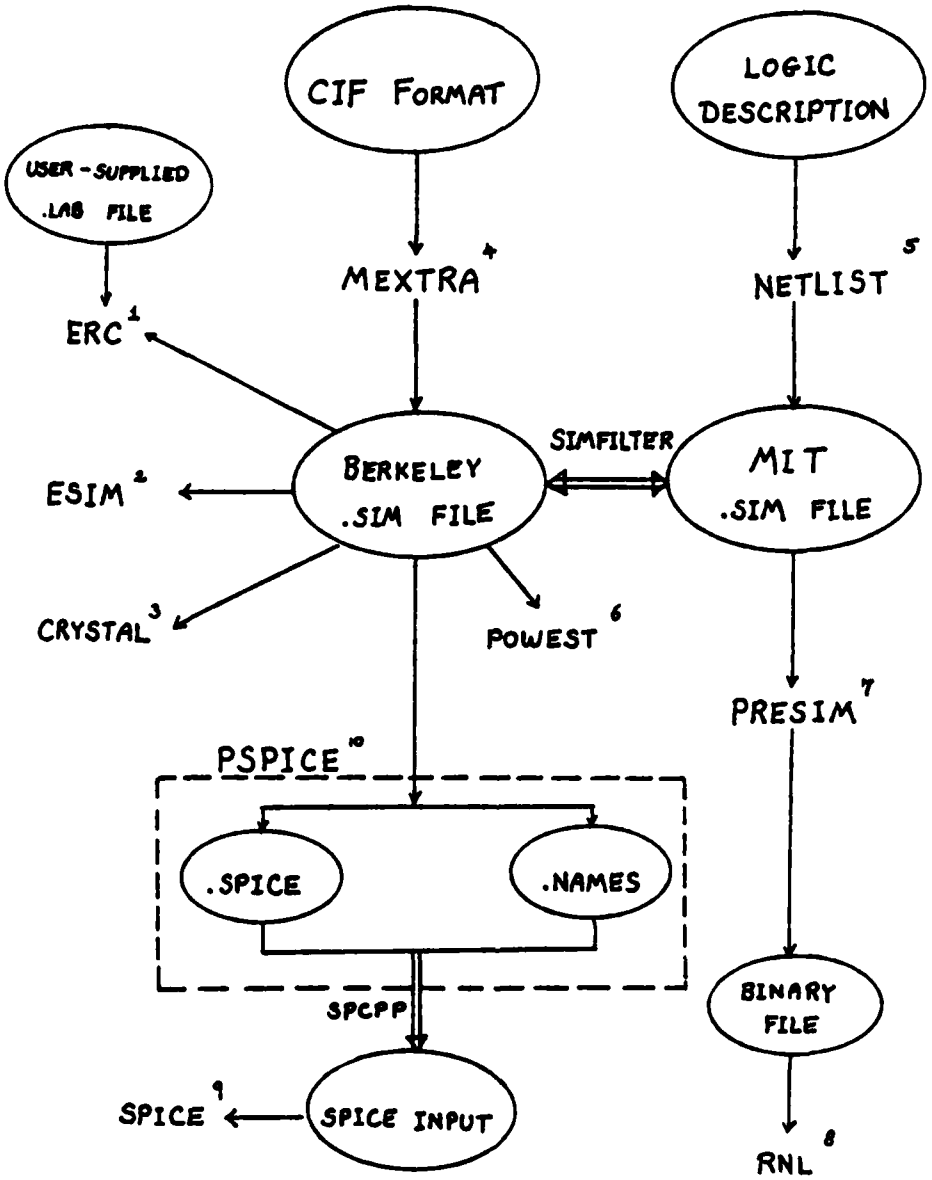
2.0 OVERVIEW OF VLSI DESIGN TOOLS

LAYOUT TOOLS : FUNCTIONAL CHART



1. Procedural approach to layout generation
2. Graphical approach to layout generation
3. Another PLA generator besides PLAP
4. Library of subroutines
5. Pieces rectangular cells together in an array
6. Design rules checker for NMOS, CMOS, or other
7. Design rules checker for NMOS

SIMULATION TOOLS : FUNCTIONAL CHART



1. Electrical Rules Checker
2. Event-driven switch level simulator
3. VLSI timing analyzer
4. Circuit extractor
5. Network description to .SIM file
6. Power Estimation
7. Preprocessor for RNL
8. Timing and logic simulator
9. Circuit simulator
10. Shell script to prepare input files for SPICE

TOOL DESCRIPTIONS

The following is a brief overview of the vlsi-tools we are using. An asterisk (*) appears after the name of tools that are not distributed on this tape. A dagger (†) refers to tools from the above-mentioned sources for which UW/NW VLSI Consortium modifications are available.

The functonal design tools translate high level design descriptions into layout tool input. Layout tools are used to design the actual artwork for the circuit; display tools are used to display circuit designs. Electrical and design rule checkers are included as well as a variety of timing and logic simulation tools.

Functional Design Tools

Translate high level design descriptions into layout tool input.

peg * Translates a language description of a finite state machine into logic equations compatible with *eqntott*. (Gordon Hamachi, UCB)

eqntott * Converts logic equations into a truth table format to be used as input to *mkpla*, *tpla* or *plap*. (Bob Cmelik, UCB)

Layout Tools

Design artwork for circuit.

plap Compiles a pascal design file created by the user, links it to a package of artwork generation routines, and executes it to produce artwork data files in an internal database format ("db"). Handles general nonmanhattan designs. (Bruce Yanagida, Boeing)

caesar * A powerful display editor for manhattan designs written at Berkeley. Runs on AED512 or Metheus Omega 440 color displays. Requires a design to be inputted in *caesar* format but will optionally output in CIF. (John Ousterhout, UCB)

tpack * A collection of 'C' routines that assembles *caesar*-generated artwork tiles into semi-regular design modules. This is not an executable program but a library of routines. Employed by *quilt* and *tpla*. (Robert Mayo and John Ousterhout, UCB)

quilt * Assembles *caesar*-generated artwork tiles into a rectangular array. (Robert Mayo, UCB)

tpla * Technology independent artwork generator. Employs *caesar*-generated artwork tiles and a truth table. (Robert Mayo, UCB)

mkpla * Generates nmos pla artwork from a truth table. This program has been largely superceded by *tpla*. (Howard A. Landman, UCB)

Display Tools

Display circuit designs.

cifplot † Berkeley program that plots a CIF design in stipple patterns on a versatec printer. Modified at UW to run on a Printronix. Has an option to extract the electrical circuit (the .sim file) for simulation. (Dan Fitzpatrick, UCB)

penplot Penplotting programs for HP7221 and HP7580 pen plotters. Compatible with either "db" or CIF formatted designs. (Boeing)

dbdi Displays a "db" formatted design on an AED512 color display.

Rule Checkers

Geometric and electrical rule checking.

alldrc Design rule checker from Carnegie-Mellon. Checks Mead and Conway nmos rules on Manhattan CIF designs. (Dorothea Haken, CMU)

fastdrc Abbreviated version of *alldrc*. (Dorothea Haken, CMU)

drcscript Merges the design rule violation files created by *alldrc* and *fastdrc* with the CIF design file for display purposes. (Dorothea Haken, CMU)

lyra * Performs hierarchical design rule check on a *caesar* formatted design using a corner based algorithm. Nmos, cmos and user-defined rulesets are available. (Michael Arnold, UCB)

erc Checks circuit description of a design for consistent electrical properties. Circuit description (the .sim file) may be obtained from the CIF file by the use of *mextra* (manhattan designs only) or *cifplot* (general nonmanhattan designs). (Boeing)

Circuit Extractors

Extract simulation database from layout database.

mextra † Extracts a Berkeley format '.sim ' file from a CIF input file. Mextra provides input files for *erc*, *esim* , *crystal* and *spice*. (Note: *Rnl* uses M.I.T. format (Dan Fitzpatrick, UCB)

Simulation Tools

Logic and timing simulation.

spice2g6 This is the well known device level circuit simulator. (Lawrence Nagel, UCB)

rnl This is an event driven "timing" simulator. It uses logic levels and a simplified circuit model to estimate timing delays through digital circuits. It also has a mode that allows it to be used as a switch (gate) level simulator. (Chris Terman, MIT)

esim * This is a gate level simulator. It uses logic levels and models transistors as perfect switches. (Chris Terman, MIT)

crystal * Crystal is a static timing verifier. It uses a simplified circuit model to estimate the worst case delay through a circuit. (John Ousterhout, UCB)

powest This reads a circuit discription in the format required by *esim* and writes estimates of its DC power requirements.

Filters and Utilities

Filters to convert from one database to another and useful utilities.

db2cif Converts from the "db" format of *plap* to CIF format. (Boeing)

cif2ca Converts from the CIF format to *caesar* format. (Peter Kessler, UCB)

ca2db Converts from *caesar* format to the "db" format of *plap*. (Bruce Yanagida, Boeing)

simfilter This is a filter that converts Berkeley style .sim files such as produced by *mextra* to MIT style .sim files used as inputs by programs such as *presim*. It also can be used to produce Berkeley style .sim files from the MIT format allowing circuits described with *netlist* to be analyzed by *spice2g6* or run through the timing verifier *crystal*. This file is acceptable input for *sim2spice* as modified by UW/NW VLSI. (Rob Daasch, UW/NW VLSI Consortium)

netlist This is a program for generating circuit descriptions (net lists). The output is an MIT style .sim file. (Chris Terman, MIT)

presim This program converts MIT style .sim files into the binary format required by *rnl*. In the process *presim* simplifies the circuit by identifying gates in the circuit. (Chris Terman, MIT)

pspice is a *csh* script that runs *sim2spice* and *spcpp*. In addition to running those programs it concatenates various files so as to create a complete *spice* input deck. This reduces considerably the effort of simulating variations on a particular circuit. (Rob Fowler, UW/NW VLSI Consortium)

sim2spice † This reads a .sim (Berkeley style) file containing a description of a circuit. It writes a .names file and a .spice file. The former contains a translation file from node names in the .sim file to the *SPICE* node numbers. The .spice file contains a description of the devices of in the circuit in a form acceptible to *SPICE*. (Dan Fitzpatrick, UCB)

spcpp is a pre-processor to facilitate the writing of *SPICE* input by allowing the user to refer circuit nodes using mnemonic node labels rather than *SPICE* node numbers. Because *Spcpp* can use the .names file produced by *sim2spice* as the translation table from labels to numbers. (Rob Fowler, UW/NW VLSI Consortium)

spice is a *csh* script for running *spice2g6*. The usage is:

spice *infile outfile*

The input file *infile* is converted to upper case, and is piped to *spice2g6*. The output of *spice2g6* is piped to *forfor* and is written in file *outfile*. (Lawrence Nagel, UCB)

NAME

ca2db – convert a Caesar cell to PLAP compatible data base files.

SYNOPSIS

ca2db [*options*] *symbolname*

DESCRIPTION

Ca2db converts any cell created by Caesar and all its children to PLAP compatible data base format. That is, it creates the **.sym** and **.att** files for the cells so they may be called from a PLAP program.

To facilitate the ease of interconnecting a Caesar created cell using PLAP, Caesar labels are also translated into attributes in the PLAP **.att** file. Each Caesar label produces two PLAP attributes:

labelname_x The *x* location of *labelname*

labelname_y The *y* location of *labelname*

Therefore, if the ports of a Caesar cell are labeled, it is trivial to use the PLAP fetch attribute command (**fa**) to get the port locations.

The *options* are as follows:

–l *lambda*
lambda = integer number of centimicrons per lambda in the design. Default is 200.

–r *resolution*
resolution = integer number of units per lambda in the PLAP **.sym** file. Default is 20.

–f *directory*
directory = directory name where the Caesar files reside. Default is the current directory.

–t *directory*
directory = directory name where the PLAP files should be written. Default is the current directory.

Note: *symbolname* should *not* have the **.ca** extension.

DIAGNOSTICS

If an error occurs, a message is written to standard error and the program exits with a non-zero status. *Ca2db* assumes the Caesar data base is intact, so don't mess with those files!

FILES

symbolname.ca
symbolname.att
symbolname.sym

SEE ALSO

caesar(cad1)
plap(1.vlsi), penplot(1.vlsi)

***PLAP User's Guide*, VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington.

Editing VLSI Circuits with Caesar, John Ousterhout, University of California, Berkeley.

AUTHOR

Bruce A. Yanagida

NAME

cifplot – CIF interpreter and plotter for displaying VLSI circuits

SYNOPSIS

cifplot [*options*] *file1*.**cif** [*file2*.**cif** ...]

DESCRIPTION

Cifplot takes a description in Cal-Tech Intermediate Form (CIF) and produces a plot. CIF is a low-level graphics language suitable for describing integrated circuit layouts. Although CIF can be used for other graphics applications, for ease of discussion it will be assumed that CIF is used to describe integrated circuit designs. *Cifplot* interprets any legal CIF 2.0 description including symbol renaming and Delete Definition commands. In addition, a number of local extensions have been added to CIF, including text on plots and include files. These are discussed later. Care has been taken to avoid any arbitrary restrictions on the CIF programs that can be plotted.

To get a plot call *cifplot* with the name of the CIF file to be plotted. If the CIF description is divided among several files call *cifplot* with the names of all files to be used. *Cifplot* reads the CIF description from the files in the order that they appear on the command line. Therefore the CIF *End* command should be only in the last file since *cifplot* ignores everything after the *End* command. After reading the CIF description but before plotting, *cifplot* will print a estimate of the size of the plot and then ask if it should continue to produce a plot. Type **y** to proceed and **n** to abort. A typical run might look as follows:

```
% cifplot lib.cif sorter.cif
Window -5700 174000 -76500 168900
Scale: 1 micron is 0.004075 inches
The plot will be 0.610833 feet
Do you want a plot? y
```

After typing **y** *cifplot* will produce a plot on the Benson-Varian plotter.

Cifplot recognizes several command line options. These can be used to change the size and scale of the plot, change default plot options, and to select the output device. Several options may be selected. A dash(-) must precede each option specifier. The following is a list of options that may be included on the command line:

-w *xmin xmax ymin ymax*

(**window**) This option specifies the window; by default the window is set to be large enough to contain the entire plot. The windowing commands lets you plot just a small section of your chip, enabling you to see it in better detail. *Xmin*, *xmax*, *ymin*, and *ymax* should be specified in CIF coordinates.

-s *float*

(**scale**) This option sets the scale of the plot. By default the scale is set so that the window will fill the whole page. *Float* is a floating point number specifying the number of inches which represents 1 micron. A recommended size is 0.02.

-l *layerlist*

(**layer**) Normally all layers are plotted. This option specifies which layers NOT to plot. The *layerlist* consists of the layer names separated by commas, no spaces. There are some reserved names: **allText**, **bbox**, **outline**, **text**, **pointName**, and **symbolName**. Including the layer name **allText** in the list suppresses the plotting of text; **bbox** suppresses the bounding box around symbols. **outline** suppresses the thin outline that borders each layer. The keywords **text**, **pointName**, and **symbolName** suppress the plotting of certain text created by local extension commands. **text** eliminates text created by user extension 2. **pointName** eliminates text created by user extension 94. **symbolName** eliminates text created by user extension 9. **allText**, **pointName**, and

symbolName may be abbreviated by **at**, **pn**, and **sn** repectively.

-c *n* (**copies**) Makes *n* copies of the plot. Works only for the Varian and Versatec. Default is 1 copy.

-d *n* (**depth**) This option lets you limit the amount of detail plotted in a hierarchically designed chip. It will only instanciate the plot down *n* levels of calls. Sometimes too much detail can hide important features in a circuit.

-g *n* (**grid**) Draw a grid over the plot with spacing every *n* CIF units.

-h (**half**) Plot at half normal resolution. *(Not yet implemented.)*

-e (**extensions**) Accept only standard CIF. User extensions produce warnings.

-I (**non-Interactive**) Do not ask for confirmation. Always plot.

-L (**List**) Produce a listing of the CIF file on standard output as it is parsed. Not recommended unless debugging hand-coded CIF since CIF code can be rather long.

-a *n* (**approximate**) Approximate a roundflash with an *n*-sided polygon. By default *n* equals 8. (I.e. roundflashes are approximated by octagons.) If *n* equals 0 then output circles for roundflashes. (It is best not to use full circles since they significantly slow down plotting.) *(Full circles not yet implemented.)*

-b *"text"*
(**banner**) Print the text at the top of the plot.

-C (**Comments**) Treat comments as though they were spaces. Sometimes CIF files created at other universities will have several errors due to syntactically incorrect comments. (I.e. the comments may appear in the middle of a CIF command or the comment does not end with a semi-colon.) Of course, CIF files should not have any errors and these comment related errors must be fixed before transmitting the file for fabrication. But many times fixing these errors seems to be more trouble than it is worth, especially if you just want to get a plot. This option is useful in getting rid of many of these comment related syntax errors.

-r (**rotate**) Rotate the plot 90 degrees.

-N (**Printronix**) Send output to the Printronix.

-V (**Varian**) Send output to the Varian. (This is the default option.)

-W (**Wide**) Send output directly to the Versatec.

-S (**Spool**) Store the output in a temporary file then dump the output quickly onto the Versatec. Makes nice crisp plots; also takes up a lot of disk space.

-T (**Terminal**) Send output to the terminal. (Not yet fully implemented.)

-Gh
-Ga (**Graphics terminal**) Send output to terminal using it's graphics capablities. **-Gh** indicates that the terminal is an HP2648. **-Ga** indicates that the terminal is an AED 512.

-X *basename*
(**eXtractor**) From the CIF file create a circuit description suitable for switch level simulation. It creates two files: *basename*.**sim** which contains the circuit description, and *basename*.**node** which contains the node numbers and their location used in the circuit description.

When this option is invoked no plot is made. Therefore it is advisable not to use any of the other options that deal only with plotting. However, the -w, -l, and -a options are still appropriate. To get a plot of the circuit with the node numbers call *cifplot* again, without the **-X** option, and include *basename*.**nodes** in the list of CIF files to be

plotted. (This file must appear in the list of files before the file with the CIF End command.)

-c *n* (**copies**) This option specifies the number of copies of the plot you would like. This allows you to get many copies of a plot with no extra computation.

-P *patternfile*

(**Pattern**) This option lets you specify your own layers and stipple patterns. *Patternfile* may contain an arbitrary number of layer descriptors. A layer descriptor is the layer name in double quotes, followed by 8 integers. Each integer specifies 32 bits where ones are black and zeroes are white. Thus the 8 integers specify a 32 by 8 bit stipple pattern. The integers may be in decimal, octal, or hex. Hex numbers start with '0x'; octal numbers start with '0'. The CIF syntax requires that layer names be made up of only uppercase letters and digits, and not longer than four characters. The following is example of a layer description for poly-silicon:

```
"NP"  0x08080808 0x04040404 0x02020202 0x01010101
      0x80808080 0x40404040 0x20202020 0x10101010
```

-F *fontfilename*

(**Font**) This option indicates which font you want for your text. The *fontfilename* must be in the directory */usr/lib/vfont*. The default font is Roman 6 point. Obviously, this option is only useful if you have text on your plot.

-O *filename*

(**Output**) After parsing the CIF files, store an equivalent but easy to parse CIF description in the specified file. This option removes the include and array commands (see next section) and replaces them with equivalent standard CIF statements. The resulting file is suitable for transmission to other facilities for fabrication.

In the definition of CIF provisions were made for local extensions. All extension commands begin with a number. Part of the purpose of these extensions is to test what features would be suitable to include as part of the standard language. But it is important to realize that these extensions are not standard CIF and that many programs interpreting CIF do not recognize them. If you use these extensions it is advisable to create another CIF file using the **-O** options described above before submitting your circuit for fabrication. The following is a list of extensions recognized by *cifplot*.

0I *filename*;

(**Include**) Read from the specified file as though it appeared in place of this command. Include files can be nested up to 6 deep.

0A *s m n dx dy* ;

(**Array**) Repeat symbol *s* *m* times with *dx* spacing in the x-direction and *n* times with *dy* spacing in the y-direction. *s*, *m*, and *n* are unsigned integers. *dx* and *dy* are signed integers in CIF units.

1 *message*;

(**Print**) Print out the message on standard output when it is read.

2 *"text" transform* ;

2C *"text" transform* ;

(**Text on Plot**) *Text* is placed on the plot at the position specified by the transformation. The allowed transformations are the same as the those allowed for the Call command. The transformation affects only the point at which the beginning of the text is to appear. The text is always plotted horizontally, thus the mirror and rotate transformations are not really of much use. Normally text is placed above and to the right of the reference point. The **2C** command centers the text about the reference point.

9 *name*;

(**Name symbol**) *name* is associated with the current symbol.

94 *name x y*;

94 *name x y layer*;

(**Name point**) *name* is associated with the point (*x*, *y*). Any mask geometry crossing this point is also associated with *name*. If *layer* is present then just geometry crossing the point on that layer is associated with *name*. For plotting this command is similar to text on plot. When doing circuit extraction this command is used to give an explicit name to a node. *Name* must not have any spaces in it, and it should not be a number.

FILES

~cad/.cadrc
~/.cadrc
~cad/bin/vdump
/usr/lib/vfont/R.6
/usr/tmp/#cif*

SEE ALSO

cadrc(cad5)

A Guide to LSI Implementation, Hon and Sequin, Second Edition (Xerox PARC, 1980) for a description of CIF.

AUTHOR

Dan Fitzpatrick (UCB)

MODIFICATIONS

(UW/NW VLSI Consortium, University of Washington)

BUGS

The **-r** is somewhat kludgy and does not work well with the other options. Space before semi-colons in local extensions can cause syntax errors.

The **-O** option produces simple cif with no scale factors in the DS commands. Because of this you must supply a scale factor to some programs, such as the **-l** option to *cif2ca*.

NAME

db2cif – converts plap layout database files into CIF

SYNOPSIS

db2cif [–i]

DESCRIPTION

Db2cif creates a complete CIF file for a specified symbol in a PLAP database. Upon invocation, *db2cif* will prompt for the PLAP database input file (extensions are not necessary and the tilda character is allowed). The response to this should be the pathname of the *symbolname* for which the CIF file is desired. If the *plap* symbol library, *libraryname* is in the current directory, then the response should be *libraryname/symbolname*.

input file ? *libraryname/symbolname*

The output filename is *symbolname*.**cif** and is created in the current working directory.

If a symbol is out of date (it contains other symbols that have been created more recently) an error message is produced and after the entire database has been checked, aborts without creating a CIF file (see –i option below).

The options are:

–i Specifies that a CIF file is to be created in spite of any out-of-date symbol errors. Error messages will not be suppressed and a comment to the effect that out-of-date symbols were encountered will be included in the CIF output.

FILES

libraryname/symbolname.att
libraryname/symbolname.sym
symbolname.cif

SEE ALSO

plap(1.vlsi)
cifplot(1.vlsi), mextra(1.vlsi), alldrc(1.vlsi), fastdrc(1.vlsi)

AUTHOR

Bruce A. Yanagida (Boeing Aerospace)

BUGS

Db2cif will not work if executed within the *libraryname* directory. *Libraryname* is required in the input file pathname and therefore must *not* be the current directory.

Options may not be combined on the command line.

NAME

dbdi – *interactive color graphics display program for VLSI designs*

SYNOPSIS

dbdi

DESCRIPTION

Dbdi is a color graphics display program that displays PLAP-generated database files.

Dbdi first prompts for a datafile name. This name must be of the form *dblibname/basename* where *dblibname/basename*.**att** and *dblibname/basename*.**sym** exist. *Dblibname* is a pathname to the PLAP-generated database files, even if located in the current working directory (see BUGS below). The program then parses the files and loads them into memory. At this point the program will search the current directory for a file named ".dl" and will execute the *dbdi* commands contained therein. This allows the user to set up the initial startup conditions.

The program then prompts for the interactive commands summarized below:

n *symbolname*
: Selects the symbol for subsequent plotting. Use "**m**" command for a listing of the symbols in the file.

e *symbolnamelist*
: Selects the symbol(s) to be excluded from the next plot. Symbol names should be separated by spaces and terminated by a carriage return.

l *layernamelist*
: Selects the layer(s) to be displayed in the next plot. If no layername is given a list of possible layers if returned.

s
: Lists the names of the symbols contained in the current CIF file.

g *n*
: Sets the new grid size to be *n* lambda by *n* lambda.

w *x1 y1 x2 y2*
: Sets new lower-left (*x1,y1*) and upper-right (*x2,y2*) window coordinates. If no parameters are given, the current lower-left and upper-right window coordinates are returned.

showwindow
: Reports the current plot port display size in lambda units

9
: Toggles plotting of labels on or off.

m
: Shows the current selections in effect.

p *option*
: Plots the current windowing of the current symbol with current selections in effect. If the **noclear** option is given then the display is not cleared before the new plot. After the plotting of the symbol, all selections except the symbol to be plotted are cleared and a new display specification is required for subsequent uses of **p**.

backp *n*
: Plots from the plot queue, where *n* is :
  0 replot the current (top of plot queue) plot (ie., replot)
  1 replot the previous plot (default)
  2 replot the penultimate plot
  Note: the plot queue is unchanged

savep *n*
: Saves current plot (top of plot queue) in user buffer *n*. Currently there are 3 user plot buffers (1-3) available. (plot buffer 1 is the default)
  Note: the plot queue is unchanged

restorep *n*
: Replots the plot saved in user buffer *n*. (plot buffer 1 is the default)
  Note: the plot queue is unchanged

dig Enters digitizer mode. The following options for digitizer mode are displayed in the prompt area:

zoom
unzm
pan
pan1
exit

The bit-pad buttons have the following functions when the cursor is in the plot area:

0 report cursor position in lambda coordinates
1 adjust lower left origin of selection window
2 exit the current command
3 adjust size of selection window

set *param val*

Sets the designated **dbdi** program parameter to the given value(s). Currently available parameters are:

tech the program's technology and color map files. This parameter option requires 2 fully qualified filenames for *.tec* and *.cmp*.

default nmos*.tec* & nmos*.cmp* (*val* is not explicitly specified). These files must be located in a directory specified in the user's search path.

{< /r} *filename*

Reads and executes the given **dbdi** command file. The default file type is *.dis*.

pausescript

Temporarily discontinues input from the script file. This can only be meaningful if the command is within the body of a script file. Input from the script file can be resumed or terminated.
(see **resumescript** and **endscript**)

resumescript

Resumes input from a paused script file.
(see **pausescript**)

endscript Closes and discontinues input from a paused script file.
(see **pausescript**)

echo Echos a message to the output file. This can only be meaningful if the command is within the body of a script file.

echowait Echos a message to the output line and await user acknowledgement before continuing. This can only be meaningful if the command is within the body of a script file.

interrupt Stops the program and prompts to be restarted. Upon restart, the display monitor will be reset and the screen will be erased. This can only be meaningful if the command is within the body of a script file.
Note: stopping during ploting may cause a plot command to be terminated in the middle, which can cause a slightly erroneous display.

reset Resets the display monitor; the current symbol and selected options are unchanged

q Quits the program; this command can only be given after a plot has been displayed.

FILES

dblibname/*basename*.cif
dblibname/*basename*.att
dblibname/*basename*.sym

technology.tec (see *tech(5.vlsi)* for *file* configuration)
technology.cmp (see *tech(5.vlsi)* for *file* configuration)

SEE ALSO

plap(1.vlsi), penplot(1.vlsi), cifplot(1.vlsi), db2cif(1.vlsi)
tech(5.vlsi)

AUTHOR

(Boeing Aerospace)

DIAGNOSTICS

All user input is checked for consistency and syntactic correctness and an explanation of the incorrectness or inappropriateness of the input is given to the user in a cryptic fashion.

BUGS

The data file name must include some pathname syntax, even if redundant (eg., *./filename* rather than just *filename*).

NAME

fastdrc, alldrc - design rule checker (vlsi-cad simulation tool)

SYNOPSIS

fastdrc [**-ku**] *basename* [*lambda*]
alldrc [**-ku**] *basename* [*lambda*]
drcscript *basename*.cif

DESCRIPTION

The *drc* analyzes an nMOS CIF file for geometric rule violations, using Mead/Conway rules. All of these rules are checked by *alldrc* , some of the less critical ones are ignored by *fastdrc*. It is limited to rectilinear, orthogonal geometry. Wires are taken apart into rectangles, and round flashes are approximated by squares. Polygons and non-manhattan rectangles are simply ignored.

The options are as follows:

-k Keep around all intermediate files.

-u Keep around files of unfiltered error messages.

lambda Specify new lambda to be *lambda* microns. Default is 2.5 microns.

For large files, *drc* should be run in batch mode (an 8000 transistor chip takes 2.4 11/780 cpu hours).

Drc ouput files must be plotted or viewed graphically.

When *drc* finds violations, it creates CIF files of rectangles marking the geometric edges involved. These markers are placed on the glass layer. Separate files are created for each class of error, named err.*errortype*.*basename*.

A shell script will merge these separate files with the original design CIF file and add labels indicating the error type. This maay be accomplished by using the command

drcscript *basename*.cif

Drcscript creates the file drc*basename*.clf which can be viewed with *dbdi* and *caesar*. Before *caesar* is invoked the filter program *cif2ca* should be run on drc*basename*.clf. Errors show up as orange boxes in the glass layer. Each pair of boxes involved in an error will have an associated *errortype* label which will be located at the midpoint between the centers of the two boxes.

nMOS rules checked by fastdrc:

Abbrev	Rule	Lambda
dS	diffusion spacing	3.0
iOg	implant-gate overlap	1.5
iSg	implant-gate spacing	2.0
pS	poly spacing	2.0
pOg	poly-gate overlap	2.0
pSd	poly-diff spacing	1.0
cS	cut-cut spacing	2.0
dcSg	diff-cut to gate 2.0	
mW	metal width	3.0

Additional nMOS rules checked by alldrc:

Abbrev	Rule	Lambda
iNOg	implants with no gates	
XC	cuts with no D or P	
dW	diffusion width	2.0

ntdW	non-xtr diff width	2.0	
iS	implant-implant		1.5
pW	poly width	2.0	
gW	gate width	2.0	
cW	cut min width	2.0	
cL	cut max length	6.0	
mOc	metal-cut overlap	1.0	
dOc	diff-cut overlap	1.0	
pOc	poly-cut overlap	1.0	

SEE ALSO

dbdi(1.vlsi), caesar(cad1), cif2ca(1.vlsi), penplot(1.vlsi)

A Geometric Design Rule Checker, Dorothea Haken, VLSI Document V053, Carnegie Mellon, 9 June 1980.

FILES

basename.cif
err*.errortype.basename*
drc*basename*.cif
err*basename*.cif

AUTHOR

Dorothea Haken (CMU)

BUGS

The poly-overlap-gate check fails when the overlap is exactly zero.

Spacing checks do not consider mutual connectivity. Sometimes wierd things will happen near butting contacts.

Cuts in diffusion or poly that do not have metal covering are not reported.

Diagonal spacing checks do not consider the true diagonal distance.

To abort *drc*, hit the <*break*> key and wait while it outputs some error messages until it eventually quits.

NAME

erc – VLSI (nMOS) electrical rules checker

SYNOPSIS

erc [**-abcdeghijmnrstvw**] [**-l** *float*] [**-u** *n*] *basename*
ercscript *basename*.cif

DESCRIPTION

Erc reads a switch level circuit description of an nMOS chip produced by *mextra* or *cifplot -X* and a designer supplied list of inputs and outputs, and produces a listing of possible electrical rule violations, a cross-reference of nodes and transistors, and CIF code to display the locations of the electrical rules violations with *dbdi*, *cifplot*, *penplot* or *caesar*. (Before *caesar* is invoked the filter program *cif2ca* must be run on the *erc* cif output file.)

Erc will read the files *basename*.**sim**, *basename*.**lab**, *basename*.**nodes**, *basename*.**al** and create up to five new files: *basename*.**erc**, *basename*.**crf**, *basename*.**err**, **erc***basename*.**cif** and **erc***basename*.**err**. If *basename* is greater than seven characters in length only the first seven characters will be used for *basename* in the files **erc***basename*.**err** and **erc***basename*.**cif**.

Erc recognizes many command line options. These can be used to enable or disable certain tests and to specify whether a cross-reference will be generated. Several options may be selected. A summary of the options in effect is placed at the beginning of the file *basename*.**erc**. A dash (-) may precede each option specifier or options may be concatenated together with a single leading dash. The following is a list of options that may be included on the command line:

-a Disables checking for transistors with the same source and drain node.

-b Disables the pass transistor gating pass transistor check.

-c This option causes the file *basename*.**crf** to be generated. This file contains a cross-reference of all transistors and nodes. The format of this file is discussed later.

-d Causes all depletion mode transistors that are not attached to *Vdd* to be listed.

-e This option causes all enhancement mode transistors attached to *Vdd* to be listed.

-g The transistor reachable from ground test is disabled.

-h Disables listing of badly formed transistors.

-i Disables listing of transistors gated by *GND*.

-j Disables listing of transistors gated by *Vdd*.

-l *float* A floating point value is given. This becomes the lower limit for the k-ratio test. Default is 4.000.

-m Disables multiple pullup transistors on the same node test.

-n Causes explicit listing of the locations of all unconnected nodes instead of giving the count only.

-r Disables ratio checking.

-s Disables listing of *Vdd-GND* shorts across transistors.

-t Causes explicit listing of the locations of all unconnected transistors instead of giving the count only.

-u *n* The integer *n* specifies cif units per lambda (250 is the default).

-v This option disables the reachable from *Vdd* test.

-w Disables the pullups separated by only 1 pass transistor test.

INPUT FILES

basename.**sim**

basename.**al**

basename.**nodes**

These files are obtained by running *mextra* or by running *cifplot* with the *-X* option. No modifications to these files are required for *erc*. It is important to note that *erc* understands only *Vdd* for power and *GND* for ground.

basename.**lab**

The designer must inform *erc* of his circuit's inputs and outputs with the *basename*.**lab** file. There is one entry in *basename*.**lab** for each input and output of the circuit being checked. The format for a single entry is as follows:

[non-restored |restored] input |output |bidirectional [pad] *nodename*

The arguments enclosed by [] are optional. Alternatives are separated by '|'. *Nodename* is the name of the input or output.

OUTPUT FILES

basename.**erc**

This file contains a summary of the tests *erc* applied to the circuit, a breakdown of the number of transistor types, and warnings of any potential errors *erc* found.

The first page summarizes the tests applied to the circuit and has a count of transistors acting as pullups, pulldowns, precharges, lightning arrestors, internal pulldown and pass transistors, and transistors that are errors by themselves. After this first page any errors that *erc* detected are listed. Each different type of error starts a new page of the output.

basename.**crf**

Basename.**crf** contains a cross-reference for all transistors and nodes. *Basename*.**crf** is generated with the **-c** option.

The first section has a numbered entry for each transistor in *basename*.**sim**. The entry consists of type (enhancement or depletion), gate, source, and drain nodes, length and width of the channel in lambda, and the x and y coordinates in lambda.

Example of a transistor entry:

[1] e latch 67 Vdd 2.0 2.0 9.0 17.5

This is transistor #1. It is enhancement mode. Its gate, source and drain nodes are *latch*, *67* and *Vdd* respectively. This transistor is 2 lambda by 2 lambda and is located at (9.0,17.5) relative to the origin.

The second section of *basename*.**crf** has an entry for each node in the circuit. An entry consists of the node's name, its x and y coordinate, input or output type, a list of transistor gates the node is attached to, and a list of transistor sources and drains that this node is connected to.

Example of a node entry:

[SR.in 14.5 4.0i] G: 6 S&D: 17 12

This node's name is *SR.in*. (14.5,4.0) is a point located on this node. The *i* after the x and y coordinates means this is an input. When the characters *b*, *i*, *n*, and *o* are in this field they mean *bidirectional*, *input*, *none* (not an *input* or *output*), and *output* respectively. After *G:* there is a list of transistor numbers that are gated by this node(there may be none). After *S&D:* there is a list of transistors that have their source or drain attached to this node. The transistor numbers in the node entries correspond to the

transistor numbers in the first section of this file.

erc *basename*.**clf**

This file should be used with *dbdi*, *penplot*, or *ceasar* as an aid in debugging the circuit. It contains the original CIF file together with a symbol that has glass layer boxes around possible errors in the circuit. The symbol that contains the original CIF file and the error layer is named *ercmerge* and the symbol containing the errors only is called *ercerrors*. All other symbol names remain unchanged. At the center of each error box there is a label that denotes the type of error detected. A list of these labels and the type of error they denote follows:

E_PAD	This pad is the source or drain of a pass transistor.
E_RAT	The *k*-ratio for this pullup is less than the allowed lower limit (default limit is 4.000).
E_MPU	This node has more than one pullup attached to it.
E_PPU	Two nodes with pullups are separated by one pass transistor.
E_VDD	This transistor is not reachable from *Vdd*.
E_GND	This transistor is not reachable from *GND*.
E_PUD	This pullup is not depletion mode.
E_PDE	This pulldown is not enhancement mode.
E_ERR	This transistor's nodes are nonsense.
E_PGP	This pass transistor is gated by another pass transistor.
E_SME	This transistor has the same source and drain nodes.
E_AON	This transistor is always on.
E_AOF	This transistor is always off.
E_SRT	There is a power to ground short from the source to the drain of this transistor.
E_TNC	This transistor is not connected to anything.
E_NNC	This node is not connected to anything.

erc *basename*.**err**

This file contains a CIF symbol containing only the errors *erc* found. Use this file to look at only the error boxes.

basename.**err**

This file contains the CIF code that is destined to become the symbol *ercerrors* in the file **erc** *basename*.**clf** and **erc** *basename*.**err**. This file is only meant to be used if **erc** *basename*.**clf** or **erc** *basename*.**err** has been inadvertantly removed. To recreate the merged representation of the CIF file type:

ercscript *basename*.**clf**

This will make two files, **erc** *basename*.**clf** and **erc** *basename*.**err**.

FILES

basename.sim
basename.al
basename.nodes
basename.lab
basename.erc
basename.cif
erc *basename*.cif
erc *basename*.err
basename.err

SEE ALSO

cifplot(1.vlsi), mextra(1.vlsi), drc(1.vlsi)

AUTHOR

Keith Pennick (Boeing Aerospace)

BUGS

Doesn't check k-ratios for pulldown networks over 1-level deep.

Multiple errors at the same node or transistor make the error messages diplayed by the display programs hard to read.

CIF resolution for the error symbol *ercerrors* in erc*basename*.cif is only 1 lambda. This sometimes results in a label not on any layer. The corresponding error message in *basename*.erc has the exact coordinates. This will be changed in an upcoming version.

NAME

laytools – Introduction to VLSI layout tools

DESCRIPTION

This document is an introduction to the tools in the UW/NW-VLSI Consortium collection that are used in connection with the layout of VLSI circuits. These fall into four categories:

functional design tools	which translate high level design descriptions into layout tool input;
layout tools	which are used to design the actual artwork for the circuit;
display tools	which are used to display circuit designs;
rule checkers	which are used to check geometric and electrical parameters of designs;

The functional design tools are:

peg	Translates a language description of a finite state machine into logic equations compatible with *eqntott*.
eqntott	Converts logic equations into a truth table format to be used as input to *mkpla*, *tpla* or *plap*.

The layout tools are:

plap	Compiles a pascal design file created by the user, links it to a package of artwork generation routines, and executes it to produce artwork data files in an internal database format ("db"). Handles general nonmanhattan designs.
db2cif	Converts from the "db" format of *plap* to CIF format.
cif2ca	Converts from the CIF format to *caesar* format.
caesar	The renowned graphical editor of manhattan designs written at Berkeley. Requires a design to be in *caesar* format but will optionally output in CIF.
tpack	A collection of 'C' routines that assembles *caesar*-generated artwork tiles into semi-regular design modules. This is not an executable program but a library of routines. Employed by *quilt* and *tpla*
quilt	Assembles *caesar*-generated artwork tiles into a rectangular array.
mkpla	Generates nMOS PLA artwork from a truth table. This program has been largely superceded by *tpla*.
tpla	Technology independent artwork generator. Employs *caesar*-generated artwork tiles and a truth table.

The display tools are:

cifplot	Berkeley program that plots a CIF design in stipple patterns on Versatec or Printronix dot-matrix printers. Has an option to extract the electrical circuit (the .sim file) for simulation.
penplot	Penplotting programs for HP 4 and 10 pen plotters. Compatible with either "db" or CIF formatted designs.
dbdi	Displays a "db" formatted design on an AED512 color display.

The rule checkers are:

alldrc Design rule checker from Carnegie-Mellon. Checks Mead and Conway nMOS rules on Manhattan CIF designs.

fastdrc Abbreviated version of *alldrc*.

drcscript Merges the design rule violation files created by *alldrc* and *fastdrc* with the CIF design file for display purposes.

lyra Performs hierarchical design rule check on a *caesar* formatted design using a corner based algorithm. nMOS, cMOS and user-defined rulesets are available.

erc Checks circuit description of a design for consistent electrical properties. Circuit description (the .sim file) may be obtained from the CIF file by the use of *mextra* (manhattan designs only) or *cifplot* (general nonmanhattan designs).

ercscript Merges the electrical rule violation files created by *erc* with the CIF design file for display purposes.

COMMENTS

There are two basic layout programs - *plap* and *caesar*. *Caesar* requires the availability of a graphics display such as an AED512 or Metheus Omega 440. It also restricts one to Manhattan designs. Plap has no such restrictions. Using plap to enter geometries of low level cells is somewhat tedious, however. Once a design is in either "db" or *caesar* format, conversion to CIF is done through *db2cif* or internally in *caesar*. With the design in CIF, a circuit description may be extracted using *mextra* or *cifplot* for simulation (see the "simtools" manual entry).

For more information about any of the above programs, see the appropriate manual entry.

NAME

mextra – Manhattan circuit extractor for VLSI simulation

SYNOPSIS

mextra [**-g**] [**-u** *scale*] [**-o**] *basename*

DESCRIPTION

Mextra will read the file *basename*.**cif** and create a circuit description. From this circuit description various electrical checks can be done on your circuit. The circuit description is directly compatible with *esim*, *powest*, and *erc*. There are translation programs to convert *mextra* output to acceptable *spice* input (see *sim2spice*, *pspice* and *spcpp*).

Mextra is very similar to the *cifplot* **-x** extractor, but much faster. Its average growth rate is linear in the area of the circuit and the number of rectangles, and its execution speed is an order of magnitude faster than the *cifplot* extractor. *Mextra*, however, is not as general as *cifplot*. *Mextra* handles only manhattan geometry and it can read only a subset of CIF. Yet these restrictions seem well worth the significant time advantages gained by *mextra*. *Mextra* creates four new files, *basename*.**log**, *basename*.**al**, *basename*.**sim** and *basename*.**nodes**. After *mextra* finishes it is a good idea to read the **.log** file. This contains general information about the extraction. It has a count of the number of transistors and the number of nodes. Also it contains messages about possible errors. The **.al** file is a list of aliases which can be used by *esim*. The **.nodes** file is a list of node names and their CIF locations listed in CIF format. It can be read by *cifplot* to make a plot showing the circuit with the named nodes superimposed. The form of this cifplot command is:

cifplot *basename*.**nodes** *basename*.**cif**

The **.sim** file is the circuit description for use with simulation programs and electrical rule checkers.

Names

Mextra uses the CIF label construct to implement node names and attributes. The form of the CIF label command is as follows:

94 *name x y* [*layer*];

This command attaches the label to the mask geometry on the specified layer crossing the point (x, y). If no layer is present then any geometry crossing the point is given the label.

Mextra interprets these labels as node names. These names are used to describe the extracted circuit. When no name is given to a node, a number is assigned to the node. A label may contain any ASCII character except space, tab, newline, double quote, comma, semi-colon, and parenthesis. To avoid conflict with extractor generated names, names should not be numbers or end in '*#n*' where *n* is a number.

A problem arises when two nodes are given the same name although they are not connected electrically. Sometimes we want these nodes to have the same names, other times we don't. This frequently happens when a name is specified in a cell which is repeated many times. For instance, if we define a shift register cell with the input marked 'SR.in' then when we create an 8 bit shift register we could have 8 nodes names 'SR.in'. If this happens it would appear as though all 8 of the shift register cells were shorted together. To resolve this the extractor recognizes three different types of names: *local*, *global*, and *unspecified*. Any time a local name appears on more than one node it is appended with a unique suffix of the form '*#n*' where *n* is a number. The numbers are assigned in scanline order and starting at 0. In the shift register example, the names would be 'SR.in#0' through 'SR.in#7'. Global names do not have suffixes appended to them. Thus unconnected nodes with global names will appear connected after extraction. (The **-g** causes the extractor to append unique suffixes to unconnected nodes with the same global name.) Names are made local by ending them with a sharp

sign, '#'. Names are global if they end with an exclamation mark, '!'. These terminating characters are not considered part of the name, however. Names which do not end with these characters are considered unspecified. Unspecified names are treated similar to locals. Multiple occurrences are appended with unique suffixes. By convention, unspecified names signify the designer's intention that this name is a local name, but is connected to only one node. It is illegal to have a name that is declared two different types. The extractor will complain if this is so and make the name local.

It makes no difference to the extractor if the same name is attached to the same node several times. However, if more than one name is given to a node then the extractor must choose which name it will use. Whenever two names are given to the same node the extractor will assign the name with the highest type priority, global being the highest, unspecified next, local lowest. If the names are the same type then the extractor takes the shortest name. At the end of the **.log** file the extractor lists nodes with more than one name attached. These lines start with an equal sign and are readable by *esim* so that it will understand these aliases.

Attributes

In addition to naming nodes *mextra* allows you to attach attributes to nodes. There are two types of attributes, *node attributes*, and *transistor attributes*. A node attribute is attached to a node using the CIF 94 construct, in the same way that a node name is attached. The node attribute must end in an at-sign, '@'. More than one attribute may be attached to a node. *Mextra* does not interpret these attributes other than to eliminate duplicates. For each attribute attached to a node there appears a line in the **.sim** file in the following form:

A *node attribute*

Node is the node name, and *attribute* is the attribute attached to that node with the at-sign removed.

Transistor attributes can be attached to the gate, source, or drain of a transistor. Transistor attributes must end in a dollar sign, '$'. To attach an attribute to a transistor gate the label must be placed inside the transistor gate region. To attach an attribute to a source or drain of a transistor the label must be placed on the source or drain edge of a transistor. Transistor attributes are recorded in the transistor record in the **.sim** file.

Transistors

For each transistor found by the exractor a line is added to the **.sim** file. The form of the line is:

type gate source drain length width x y
g=*attributes* **s**=*attributes* **d**=*attributes*

Type can be one of three characters, 'e' for enhancement, 'd' for depletion, or 'u' for unusual implant. (Unusual implant refers to transistors which are only partially in an implanted area. It will be necessary to write a filter to replace these transistors with the appropriate model in terms of enhacement and depletion transistors.) *Gate*, *source*, and *drain* are the gate, source, and drain nodes of the transistors. *Length and width* are the channel length and width in CIF units. *X* and *y* are the x and y coordinates of the bottom left corner of the transistor. *Attributes* is a comma seperated list of attributes. If no attribute is present for the gate, source, or drain, the **g**=, **s**=, or **d**= fields may be omitted.

The extractor guesses the length and width of a transistor by knowing the area, perimeter, and length of diffusion terminals. For rectangular transistors and butting transistors the reported length and width is accurate. For transistors with corners or for unusually shaped transistors the length and width is not as accurate.

It is possible to design a transistor with three or more diffusion terminals. The extractor considers these as *funny transistors*. They are entered in the **.sim** file in the form:

ftype gate node1 node2 ... nodeN xloc

The 'f' is followed by the type : 'e', 'd' or 'u'. *Node1 ... nodeN* are the diffusion terminal nodes. As with any circuit with 'u' transistors, any circuit with 'f' transistors must be run through a filter replacing each of the funny transistors with the appropriate model in terms of enhancement and depletion transistors.

Capacitance

The **.sim** file also has information about capacitance in the circuit. The lines containing capacitance information are of the form:

C *node1 node2 cap-value*

cap-value is the capacitance betweens the nodes in femto-farads. Capacitance values below a certain threshold are not reported. The default threshold is 50 femto-farads.

The extractor reports capacitance from two sources. Capacitance between node and substrate, and capacitance caused by poly overlapping diffusion but not forming a transistor. Transistor capacitances are not included since most of the tools that work on the **.sim** file calculate them from the width and length information.

The capacitance for each layer is calculated separately. The reported node capacitance is the total of the layer capacitances of the node. The layer capacitance is calculated by taking the area of a node on that layer and multiplying it by a constant. This is added to the product of the perimeter and a constant. The default constants are given below. Area constants are in femto-farads per square micron. Perimeter constants are femto-farads per micron.

Layer	Area	Perimeter
metal	0.03	0.0
poly	0.05	0.0
diff	0.10	0.1
poly/diff	0.40	0.0

Poly/diffusion capacitance is calculated similar to layer capacitance. The area is multiplied by constant and this is added to the perimeter multiplied by a constant. Poly/diffusion capacitance is not threshold, however.

The **-o** option supresses the calculation of capacitance, and instead, gives for each node in the circuit the area and perimeter of that node on the diffusion, poly, and metal layers. The lines containing this information look like this:

N *node diffArea diffPerim polyArea polyPerim metalArea metalPerim*

Node is the node name. *DiffArea* through *metalPerim* are the area and perimeter of the diffusion, poly, and metal layers in user defined units. (In addtion the **-o** option causes transistors with only one terminal to be recorded in the **.sim** file as a transistor with source connected to drain.)

Changing Default Values

As part of its start up procedure *mextra* reads two files: /usr/vlsibin/**.cadrc** and the **.cadrc** file in the user's home directory. *Mextra* reads these files to set up constants to be changed without recompiling. Each line in the **.cadrc** file starts with a keyword that tells the program how to process the line.

By default, *mextra* reports locations in CIF coordinates. A more convenient form of units may be specified either in the **.cadrc** file or on the command line. The form of the line in the **.cadrc** file is:

units *scale*

where *scale* is in centi-microns. The user may type in the chosen value for the scale directly.

To set units on the command line use the **-u** option.

mextra -u *scale basename*

The parameters used to compute node capacitance may be changed by including the following commands in your **.cadrc** file.

areatocap *layer value*
perimtocap *layer value*

value is atto-farads per square micron for area, and atto-farads per micron for perimeter. *layer* may be "poly", "diff", "metal", or "poly/diff".

To set the capicitor values to those given in Mead and Conway the following lines would appear in the **.cadrc** file:

```
areatocap poly 40
areatocap diff 100
areatocap metal 30
areatocap poly/diff 400
perimtocap poly 0
perimtocap diff 0
perimtocap diff 0
perimtocap metal 0
perimtocap poly/diff 0
```

The threshold for reporting capacitance may be set in the **.cadrc** file with the following line.

capthreshold *value*

A negative value sets the threshold to infinity.

Mextra knows of two technologies, nMOS and cMOS p-well. NMOS is assumed by default. To set the technology to cMOS p-well, include the following line in your **.cadrc** file:

tech cmos-pw

FILES

~/.cadrc
basename.cif
basename.al
basename.log
basename.nodes
basename.sim

SEE ALSO

powest(1.vlsi), pspice(1.vlsi), spcpp(1.vlsi), sim2spice(1.vlsi), spice(1.vlsi), drc(1.vlsi), erc(1.vlsi)
caesar(cad1),
cadrc(cad5),

AUTHOR

Dan Fitzpatrick (UCB)

MODIFICATIONS

(UW/NW VLSI Consortium, University of Washington)

BUGS

Accepts manhattan simple CIF only, use **cifplot -O** to convert complicated CIF. For unusually shaped transistors the UW/NW modified *mextra* should be used, otherwise values will be quite inaccurate. The modified *mextra* will either yield accurate values or a "reasonable" guess,

depending on the complexity of the unusual transistor. The modified *mextra* will tell you when the output values are only best estimates. The length/width ratio for unusually shaped transistors may be inaccurate. This is true for snake transistors. Attributes for funny transistors are not recorded. Node attributes are ignored unless the **-e** switch is present.

NAME

netlist – a simple network description language for VLSI circuits

SYNOPSIS

netlist *infile* [*outfile*] [**-o**] [**-s***n*] [**-d** *n,m*] [**-e** *n,m*] [**-i** *n,m*] [**-l** *n,m*] [**-p** *n,m*]

DESCRIPTION

Netlist requires an input file with any/all extensions on the command line. An optional output file can be specified. Additional options are described below;

-o Uses old input format. Size specifications are taken to be length/width rather than width/length.

-s*n* Uses number *n* as initializer for internal node names; useful when you want to merge the results of separate *net* runs.

-d *n, n* Sets the default width to *n* and length to *m* for depletion devices. The defaults are *n*=8 and *m*=2.

-e *n, n* Similar to **-d** except for enhancement devices. The defaults are *n*=2 and *m*=2.

-i *n, n* Similar to **-d** except for intrinsic devices. The defaults are *n*=2 and *m*=2.

-l *n, n* Similar to **-d** except for low-power devices. The defaults are *n*=2 and *m*=2.

-p *n, n* Similar to **-d** except for p-channel devices. The defaults are *n*=2 and *m*=2.

Netlist is a macro-based language for describing networks of sized transistors. Names in *netlist* refer to nodes, which presumably get interconnected by the user through transistors. Macros for describing transistors can be found in the *netlist* user's guide. In addition to transistor macros *netlist* provides macros that allow the user to set node capacitance, specific node delays (node delays are in 1/10ths of nanoseconds), and transistor threshold voltages. The user may also define his own macros.

SEE ALSO

simfilter(1.vlsi), presim(1.vlsi), rnl(1.vlsi)

NETLIST User's Guide, **VLSI Design Tools Reference Manual,** UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

Changes for NETLIST/PRESIM/RNL, **VLSI Design Tools Reference Manual,** UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

File Format,"" User's Guide to "sim" File Format, **VLSI Design Tools Reference Manual,** UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

AUTHOR

Christopher Terman (MIT)

NAME

penplot – display plap database or standard CIF files on a pen plotter

SYNOPSIS

penplot [–db |–cif] [–7221 |–7580]

DESCRIPTION

Penplot is a program that plots *PLAP* database files or standard CIF files on an x-y pen plotter. The current plotter options include the HP7221 and HP7580.

–db (default) Specifies that the input is from *PLAP* database files (**.att** and **.sym** files);

–cif Specifies that the input is from a standard CIF file (**.cif** file);

–7221 (default) Specifies that the plotter is the HP7221 x-y pen plotter;

–7580 Specifies that the plotter is the HP7580 x-y pen plotter.

Penplot first prompts the user for the technology to be used. Currently they are **nMOS** and **cMOS.** It then prompts for a data file name. This name must be of the form *dblibname/baseName* where *dblibname/baseName*.**cif** exists when using the **–cif** option, or *dblibname/baseName*.**att** and *dblibname/baseName*.**sym** exist when using the **–db** option. *Dblibname* is a pathname to the PLAP-generated database files.

The program can be run inside the library directory is desired.

The *penplot* commands are:

n Selects the symbol for subsequent plotting. The program prompts for the symbol name.

e Selects the symbol(s) to exclude from the next plot. The program prompts for a list of symbol names, *one at a time*, the last of which must be **"end"**. Typing an **"m"** after a symbol name causes only the minimum bounding box of the named symbol to be plotted.

l Selects the layer(s) to be displayed in the next plot. The currently available layers are listed and the program prompts for a list of layers, separated by spaces and terminated by a carriage return.

s Lists the symbol names, including bounding boxes, in the current datafile.

g Sets the new grid size to some number of lambda. The program prompts for the number of lambda per grid.

w Changes the window definition. The current window coordinates are listed and the program prompts for new lower-left and upper-right coordinates.

4 Selects 4-pen plotting capability. Default is 8-pen plotting capability. To return to 8-pen mode the program must be restarted.

9 Toggles plotting of labels on or off.

m Shows the current selections in effect.

p Plots the current symbol with current selections in effect.

? Prints the list of possible commands.

q Quits the program.

Roundflashes are approximated by 12 sided polygons.

The programs use the technology files to determine what color to plot each of the CIF layers. The following table shows the colors available and what pen stalls they must be in. Note that the spelling of the colors MUST match the spelling in the technology files. The technology files may reside anywhere in the user's search path. The first one encountered is the one used.

Stall	Color
1	black
2	blue
3	green
4	red
5	yellow, gold
6	brown
7	orange, lime-green
8	purple, violet, turquoise

Any colors not appearing in the table will be plotted as black. Furthermore, for the four pen plotters, only black, blue, green, and red are recognized with all others plotted as black.

FILES

dblibname/basename.att
dblibname/basename.sym
dblibname/basename.cif
technology.cmp (see *tech(5.vlsi)* for *file* configuration)
technology.tec (see *tech(5.vlsi)* for *file* configuration)

SEE ALSO

plap(1.vlsi), ca2db(1.vlsi)
dbdi(1.vlsi), cifplot(1.vlsi)
tech(5.vlsi)

AUTHOR

(Boeing Aerospace)

BUGS

The tilda character '~' in pathnames always gets expanded into '/users'. Don't use it.

Default values for unselected items are not given, either by the *m* command or by the command for altering those selections.

The use of *end* to terminate the exclusion list precludes the use of any symbol named end.

If 4-pen plotting capability is chosen, the program must be restarted to return to 8-pen plotting capability.

The plot interrupt capability does not work when using the HP7221 four pen plotters! For some reason the enquire/acknowledge handshaking used with those plotters screws up the interrupt processing.

NAME

plap – compile, link, and run a PLAP layout of a VLSI circuit

SYNOPSIS

plap [**–clf**] *basename*

DESCRIPTION

Plap will compile a *PLAP* IC design file, link it to the appropriate artwork modules, and run the program. A template or "shell" file which sets up the proper program environment for the IC designer is available in */usr/lib/vlsi/pshell.p* and it may be copied from any account. The designer must insert his *PLAP* code into this "shell" and rename this design file *basename*.**p**.

The only *option* available is:

-clf Runs *db2cif* on **lib**/*basename*.**sym** (see the file descriptions below) to produce the CIF file *basename*.**clf**. For this option to work, a symbol must have been defined in the *PLAP* program with its *symbolname* being the same as *basename*. The CIF file will not be produced for any other symbols defined in the *PLAP* program. Also, the library declaration in the *PLAP* program must be "**lib**".

The program creates a number of new files:

basename.**out**
an executable file is placed in the current working directory;

basename.**clf**
a standard CIF file is created in the current working directory when the **–clf** option is used.

symbolname.**att**
an attribute file for each symbol *symbolname* defined is placed in the last library declared in the design file. If no libraries are declared, a subdirectory of the current working directory named **lib** is created (if necessary) and the attribute file is placed there;

symbolname.**sym**
a modified CIF file for each symbol *symbolname* defined is placed in the last library declared in the design file. If no libraries are declared, a subdirectory of the current working directory named **lib** is created (if necessary) and the modified CIF file is placed there;

logname.**log**
a log file, which is a copy of the output to the terminal, where *logname* is the string given in the *PLAP* "initialize" command (usually this is *basename*);

The following is a summary of *PLAP* commands:

Artwork Primitives

layer(*layerName* **);**
box(*x1*, *y1*, *x2*, *y2* **);**
lbox(*layerName*, *x1*, *y1*, *x2*, *y2* **);**
wire(*width*, *xStart*, *yStart* **);** *wirePath*
lwire(*layerName*, *width*, *xStart*, *yStart* **);** *wirePath*
polygon(*xStart*, *yStart* **);** *polyPath* **[polyclose;]**

Path Primitives

xy(*xNext*, *yNext* **);**
dxy(*xDelta*, *yDelta* **);**
x(*xNext* **);**
y(*yNext* **);**
dx(*xDelta* **);**
dy(*yDelta* **);**

z(*newLayerName* **);**
w(*newWidth* **);**

Symbol Definition and Call Statements

define(*'symbolName'* **);**
endef;
draw(*'symbolName'*, *x*, *y* **);**
drawmx(*'symbolName'*, *x*, *y* **);**
drawmy(*'symbolName'*, *x*, *y* **);**
drawrot(*'symbolName'*, *x*, *y*, *vx*, *vy* **);**

Functions and Annotation Statements

lastX;
lastY;
plottext(*'textString'*, *x*, *y*, *centered* **);**
nodelabel(*'textString'*, *x*, *y*, *layerName* **);**

Boolean Switch Variables

checkswitch
manhattan
halfcheck

Macros

downroute(*nLines*, *xStartArray*, *yStart*, *xEndArray*,
yEnd, *width*, *spacing* **);**

Symbol Access Statements

library(*libraryName* **);**
withsym(*symbolName* **);**
fa(*attributeString* **);**
setatt(*attributeString*, *value* **);**

Lambda Definition

lambda(*microns* **);**

nMOS Macros

rb(*x*, *y* **);**
gb(*x*, *y* **);**
be(*x*, *y* **);**
bn(*x*, *y* **);**

bs(*x, y* **);**
bw(*x, y* **);**
pullup(*x, y, polyLength, direction* **);** *drainPath*
alpha(*'textString', x, y, scale* **);**

inpad(*padSpace* **);**
outpad(*padSpace* **);**
padtristate(*spacing* **);**

newpla(*nInputs, nMinterms, nOutputs, nFeedbacks,*
'codefileName', gutSwitch, clockSwitch **);**
getpla(*PLAname* **);**

FILES

/usr/lib/uw-vlsi/pshell.p
symbolname.att
symbolname.sym
basename.cif
basename.out
logname.log

SEE ALSO

PLAP User's Guide, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington.

AUTHOR

(Boeing Aerospace)

MODIFICATIONS

(UW/NW VLSI Consortium, University of Washington)

NAME

rnl, nl – timing and logic simulators for VLSI (nMOS) circuits

SYNOPSIS

rnl [*cmdfile*]

DESCRIPTION

Rnl (NetLisp) is a timing logic simulator for digital nMOS circuits with a lisp-like command interpreter. To use *rnl*, one needs a **.sim** file for the circuit to be simulated. This can be derived from the mask file (e.g., CIF) or developed using *netlist*, a program that processes textual schematics.

Nl is a version of *rnl* that implements a switch-level transistor model. It is called from within the *rnl* program. *Nl* is somewhat faster but does not correctly handle circuits that depend on the relative sizes of transistors for correct operation (ie., there are circuits for which *nl* and *rnl* will get different answers). However, *nl* does work for "Mead and Conway" type circuits (so called ratioless logic) -- it has been used with good results on almost all the circuits to come out of MIT in the last couple of years.

One must first convert the **.sim** file to a network file suitable for use by *rnl*. To do this run *presim*:

presim *filename*.**sim** *filename* [*config_params*]

which converts the file *filename*.**sim** into a binary file for *rnl* called *filename* (see *presim* user's guide for information on the various configuration parameters.

The optional *cmdfile* is the file *rnl* initially reads for user input. Usually one prepares a command file that reads in the network from the **.sim** file and also loads a library file of *rnl* functions. As simulation proceeds, user defined functions developed for testing the circuit can be added to the command file. Thus, initially, the command file might contain the commands

(load "nl.l")
(read-network "*filename***")**

It sill probably be necessary add some directory to "**nl.l**"; the file *filename* was prepared by *presim*. When the end-of-file is reached in the command file, input is taken from stdin. Commands and formats to be used are given in the *rnl* user's guide.

The top level of *rnl* is a simple loop:

(1) read command from current input;
(2) evaluate command, performing specified actions;
(3) print the result and loop back to (1).

The following is a list of the objects that *rnl* knows about

numbers -- signed integers. (16 bits on PDP11s, 24 bits on VAXen, 28 bits on PDP10s).
-- floating point.

strings sequences of characters enclosed in quotes ("). Useful as constants for file names, print statements, etc. Special characters can be introduced into the strings by using the backslash escapes:

'\n'	newline
'\r'	return
'\t'	tab
'\ooo'	ascii code "ooo" where ooo are octal digits

symbols variable names. Any sequence of characters that isn't a number, string, or some special character -- starting symbols with a letter, followed by more letters, numbers, and punctuation is usually a safe bet.

nodes an electrical node.

transistors a mosfet with gate, source, and drain.

lists a sequence of objects enclosed in parentheses. Standard LISP syntax applies, including dot notation. The empty list "()" is also called "nil".

subrs primitive, or built-in, functions (like +).

The functions are listed by application area. The areas are:

- arithmetic functions
- predicates
- list functions
- I/O functions
- miscellaneous functions
- special forms
- network/simulation functions
- functions defined in "nl.l"

See the *rnl* user's guide for full descriptions of the above.

FILES

nl.l (generally requires an additional path (eg. /usr/local/))

SEE ALSO

simfilter(1.vlsi), netlist(1.vlsi), presim(1.vlsi)

RNL User's Guide, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

Changes for NETLIST/PRESIM/RNL, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

File Format,"" User's Guide to "sim" File Format, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

AUTHOR

Christopher Terman (MIT)

BUGS

User defined macros with the same name as a node in the net list puts *rnl* into an infinite loop.

NAME

powest – estimate power consumption of MOS circuits

SYNOPSIS

powest [-p] [*parm=value*] ...

DESCRIPTION

Powest reads a MOS circuit description in the format required by *esim*(cad1) and writes estimates of its DC power requirements.

Two types of input lines are accepted:

d *gate source drain length width*
e *gate source drain length width*

(Other lines are discarded.) These lines specify the connectivity and size of depletion (d) and enhancement (e) MOSFET's. The three signal names *gate*, *source*, and *drain* may be composed of any non-white space characters, and may be of arbitrary length. Upper and lower case distinctions are ignored only for the signals *Vdd* and *GND*. *Length* and *width* specify the channel dimensions in (integral) CIF units (centimicrons).

Powest looks for three types of pullup transistors:

1 Depletion device with its gate connected to its source, and its drain connected to *Vdd*.

2 Depletion device with its gate not directly connected to its source, and its drain connected to *Vdd* (e.g. depletion load of the second stage of a superbuffer)

3 Enhancement device with its gate and its drain both connected to *Vdd*.

(The symmetric cases in which the sources and drains are interchanged are also recognized.)

For each of these pullup types, a count of the number of such devices is given, along with an average and maximum DC power estimate attributable to those devices. For both of these it is assumed that *Vds* equals *Vdd* (or *Vsd* equals *Vdd* in the symmetric case), and for the depletion devices, *Vgs* (*Vsg*) equals zero. In obtaining the maximum estimate it is further assumed that the devices are on all the time. For the average power estimates it is assumed that the second type of depletion pullups and the enhancement pullups are on half the time. For the first type of depletion pullups, a count, *n*, is made of the number of enhancement devices connected to a pullup's source and it is assumed that the pullup is on a fraction 1 - 2 sup -n of the time.

The user may override the default values for the supply voltage and certain process parameters used in the calculations using the *parm=value* option. The -p option causes a list of these parameters and their effective values to be printed. Values take the form of C floating point constants. The following table summarizes the accessible parameters, their default values, and their implicit units.

parm	*value*	*units*	*description*
gamma	0.4	V**.5	bulk threshold parameter
tox	9.0e-8	m	oxide thickness
u0	0.08	m**2/V-s	electron surface mobility
vsb	2.0	V	source-to-substrate voltage
vtd	-3.5	V	depletion threshold voltage
vte	0.8	V	enhancement threshold voltage
vdd	5.0	V	supply voltage

SEE ALSO

esim(cad1)

AUTHOR

Bob Cmelik (UCB)

BUGS

Pullups which don't fall into one of the three simple categories recognized can be the biggest power dissipators.

NAME

presim – a netlist preprocessor for the rnl VLSI circuit simulator

SYNOPSIS

presim *infile* [*outfile*] [*configfile*] [–c*file,min*] [–t*file,min*] [*presist,voltage*]

DESCRIPTION

Presim converts the MIT .sim file into a binary file to be used by *rnl*.

The parameters and options are as follows:

infile A net list file that must include any/all extensions;

outfile (optional) An output filename can be specified on the command line;

configfile (optional) A file to set lambda and RC parameters for nodes and transistors in the netlist (see the *presim* user's guide for descriptions of the parameters and syntax).

–c*file, min* Writes a list of node names and capacitances to the specified *file*. Only capacitances larger than *min* will be included.

–t*file, min* Writes a list of transistors and RC values to the specified *file* -- there are two entries for each transistor. The R's come from the size of the transistor, C's from the source/drain capacitance. Only RC values larger than *min* will be included.

–**presist**, *voltage*
Provides a worse-case estimate of the circuit power consumption by assuming that all the pullups (DEP or LOWP devices with drain=*Vdd*) are all on simultaneously. *Voltage* specifies the supply voltage,

SEE ALSO

simfilter(1.vlsi), netlist(1.vlsi), rnl(1.vlsi)

PRESIM User's Guide, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

Changes for NETLIST/PRESIM/RNL, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

File Format,"" User's Guide to "sim" File Format, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

AUTHOR

Christopher Terman (MIT)

NAME

pspice – prepare an input file for the Spice circuit simulator

SYNOPSIS

pspice [**-rm**] [**-nos2s**] [**-d** *defsfile*] [**-m** *modelfile*] [**-e** *expfile*] *basename*

DESCRIPTION

Pspice is a shell script for preparing *spice* input from information from several sources. *Pspice* runs *sim2spice* to convert from a *basename*.sim format circuit description to a *spice* compatible description and modifies the *sim2spice* node label translation table to be acceptable *spice* comments. It then runs *spcpp* to translate a pseudo-*spice* formatted file that contains symbolic node labels to a *spice* acceptable file. Finally, *pspice* concatenates the circuit description file, the translation table, a file of untranslated *spice* input, and the translated *spice* input into a single file named *basename*.spcin. This file is usually an acceptable *spice* input file. The optional parameters can be used to cause parts of this process to be skipped.

The options and parameters are:

-nos2s Suppresses the execution of the *sim2spice* step.

-rm Indicates that the files created in intermediate steps are to be deleted.

-d *defsfile* Specifies a file to be used as a *sim2spice* definitions file.

-m *modelfile* Specifies a file that contains *spice* input that is to be included (untranslated) in the final output. It is intended that *modelfile* name a file containing *spice* .model cards as well as other *spice* commands that are independent of the particular circuit being modeled.

-e *expfile* Specifies a file that contains pseudo-input for *spice*. *spcpp* will interpret strings in *exp* that are bracketed by '<' and '>' as node names to be translated into *spice* node numbers using the translation table (*basename*.names) created by *sim2spice*. Lines containing bracketed tokens are converted into *spice* comments. It is intended that *exp* contain *spice* commands that describe the experiment to be simulated on the circuit. The ability to use mnemonic node names makes the preparation of *spice* input much easier and it means that the description of the experiment need only be specified once, even if the circuit is modified and reextracted. If *exp* is not specified then spcpp is not executed.

basename Specifies the base name for the files describing the circuit. If *sim2spice* is run then a file named *basename*.sim must exist. If *sim2spice* is not run then the files *basename*.names and *basename*.spice must exist.

FILES

basename.sim	circuit description input to *sim2spice*
defsfile	optional *sim2spice* defs input
basename.names	modified *sim2spice* translation table output. This is read by *spcpp* (*)
basename.spice	*sim2spice* output *spice* format circuit element definitions (*)
modelfile	optional *spice* .model commands to be included in *basename*.spcin
expfile	input to *spcpp* containing pseudo-*spice* commands describing the experiment to be simulated
basename.spcx	translated output from *spcpp* (*)
basename.spcin	The *spice* input deck created by concatenating *basename*.spice, *basename*.names, *modelfile*, and *basename*.spcx

Note: Files marked (*) are deleted by the -rm option.

SEE ALSO

sim2spice(1.vlsi), spcpp(1.vlsi)
spice(1.vlsi)

mextra(1.vlsi), cifplot(1.vlsi)

AUTHOR

Robert Fowler (UW/NW VLSI Consortium, University of Washington)

DIAGNOSTICS

The error messages are intended to be self explanitory. Note that *sim2spice* and *spcpp* produce their own error messages.

BUGS

The command line is long enough to tempt a user to call *pspice* from yet another shell script. A better way to do this is to set up an alias for *pspice* with the commonly used options already set.

NAME

rnl, nl – timing and logic simulators for VLSI (nMOS) circuits

SYNOPSIS

rnl [*cmdfile*]

DESCRIPTION

Rnl (NetLisp) is a timing logic simulator for digital nMOS circuits with a lisp-like command interpreter. To use *rnl*, one needs a **.sim** file for the circuit to be simulated. This can be derived from the mask file (e.g., CIF) or developed using *netlist*, a program that processes textual schematics.

Nl is a version of *rnl* that implements a switch-level transistor model. It is called from within the *rnl* program. *Nl* is somewhat faster but does not correctly handle circuits that depend on the relative sizes of transistors for correct operation (ie., there are circuits for which *nl* and *rnl* will get different answers). However, *nl* does work for "Mead and Conway" type circuits (so called ratioless logic) -- it has been used with good results on almost all the circuits to come out of MIT in the last couple of years.

One must first convert the **.sim** file to a network file suitable for use by *rnl*. To do this run *presim*:

presim *filename*.**sim** *filename* [*config_params*]

which converts the file *filename*.**sim** into a binary file for *rnl* called *filename* (see *presim* user's guide for information on the various configuration parameters.

The optional *cmdfile* is the file *rnl* initially reads for user input. Usually one prepares a command file that reads in the network from the **.sim** file and also loads a library file of *rnl* functions. As simulation proceeds, user defined functions developed for testing the circuit can be added to the command file. Thus, initially, the command file might contain the commands

(load "nl.l")
(read-network "*filename***")**

It sill probably be necessary add some directory to "**nl.l**"; the file *filename* was prepared by *presim*. When the end-of-file is reached in the command file, input is taken from stdin. Commands and formats to be used are given in the *rnl* user's guide.

The top level of *rnl* is a simple loop:

(1) read command from current input;
(2) evaluate command, performing specified actions;
(3) print the result and loop back to (1).

The following is a list of the objects that *rnl* knows about

numbers -- signed integers. (16 bits on PDP11s, 24 bits on VAXen, 28 bits on PDP10s).
-- floating point.

strings sequences of characters enclosed in quotes ("). Useful as constants for file names, print statements, etc. Special characters can be introduced into the strings by using the backslash escapes:

'\n'	newline
'\r'	return
'\t'	tab
'\ooo'	ascii code "ooo" where ooo are octal digits

symbols variable names. Any sequence of characters that isn't a number, string, or some special character -- starting symbols with a letter, followed by more letters, numbers, and punctuation is usually a safe bet.

nodes an electrical node.

transistors a mosfet with gate, source, and drain.

lists a sequence of objects enclosed in parentheses. Standard LISP syntax applies, including dot notation. The empty list "()" is also called "nil".

subrs primitive, or built-in, functions (like +).

The functions are listed by application area. The areas are:

- arithmetic functions
- predicates
- list functions
- I/O functions
- miscellaneous functions
- special forms
- network/simulation functions
- functions defined in "nl.l"

See the *rnl* user's guide for full descriptions of the above.

FILES

nl.l (generally requires an additional path (eg. /usr/local/))

SEE ALSO

simfilter(1.vlsi), netlist(1.vlsi), presim(1.vlsi)

RNL User's Guide, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

Changes for NETLIST/PRESIM/RNL, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

File Format,"" User's Guide to "sim" File Format, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (Christopher Terman, MIT Laboratory for Computer Science).

AUTHOR

Christopher Terman (MIT)

BUGS

User defined macros with the same name as a node in the net list puts *rnl* into an infinite loop.

NAME

sim2spice – convert from mextra format to spice (circuit simulator) format

SYNOPSIS

sim2spice [**–d** *defs*] *basename*.**sim**

DESCRIPTION

Sim2spice reads the *basename*.**sim**, *basename*.**nodes** and *basename*.**al** files created by *mextra* and creates a *spice* readable circuit description, *basename*.**spice**. *Spice* requires node numbers and *sim2spice* generates a translation table *basename*.**names** which shows the *mextra* nodelabel corresponding to a given node number.

The user can specify his/her own translation table by using the *-d* option, where *defs* is a file of definitions. A definition can be used to set up equivelences between .**sim** node names and *spice* node numbers. The form of this type of definition is:

set *sim_name spice_number* [*tech*]

The *tech* field is optional. In nMOS, a special node, 'BULK', is used to represent the substrate node. For cMOS, two special nodes, 'NMOS' and 'PMOS', represent the substrate nodes for the 'n' and 'p' transistors, repectively. For example, for nMOS the .**sim** node 'GND' corresponds to spice node 0, 'Vdd' corresponds to spice node 1, and 'BULK' corresponds to spice node 2. The *defs* file for this set up would look like this:

```
set GND 0 nmos
set Vdd 2 nmos
set BULK 3 nmos
```

A definition also allows you to set a correspondence between .**sim** transistor types and and spice transistor types. The form of this definition is:

def *sim_trans spice_trans* [*tech*]

Again, the *tech* field is optional. For nMOS these definitions would look as follows:

```
def e ENMOS nmos
def d DNMOS nmos
```

Definitions may also be placed in the '.cadrc' file, but the definitions in the *defs* file overrides those in the '.cadrc' file.

The program has been extended so that a comment line beginning with "|=" is interpreted as an MIT .**sim** style node equivalence line.

To create a complete *spice* input file it is necessary to append applicable *spice* model descriptions as well as the user's *spice* simulation commands to the *basename*.**spice** file.

It is recommended in most cases that the user run *pspice* rather than *sim2spice*. *Pspice* incorporates the features of *sim2spice* but will in addition allow the user to build all of the *spice* input file in one step. *Pspice* also incorporates the features of *spcpp*.

FILES

basename.sim
basename.nodes
basename.al
basename.spice
basename.names

SEE ALSO

mextra(1.vlsi), spice(1.vlsi), pspice(1.vlsi), spcpp(1.vlsi)

AUTHOR

Dan Fitzpatrick (UCB)

MODIFICATIONS

Neil Soiffer (UCB) -- cMOS fixes.

Rob Fowler (UW/NW VLSI Consortium, University of Washington) -- node equivalence handling and misc. bug fixes.

BUGS

The only pre-defined technologies are 'nmos' and 'cmos-pw'. Only one definition file is allowed.

Warning: for nMOS circuits the node names "ENMOS" and "DNMOS" are preempted by *sim2spice* as synonyms for "BULK".

The node equivalence handling is not completely general. New nodes can be added to equivalence classes, but classes cannot be merged. This is detected and an error message is produced.

NAME

simfilter – Berkeley .sim <-> MIT .sim filter for VLSI simulators

SYNOPSIS

simfilter [–n*lambda*|–c*lambda*] *infile* [*outfile*]

DESCRIPTION

Simfilter reformats the **.sim** file format of either Berkeley or MIT (output from *mextra* for example) into the other **.sim** format. This allows layouts extracted by *mextra* or *cifplot –X* to be simulated using *rnl*. Alternatively, using the –n or –c option allows **.sim** files from *netlist* to be input for *sim2spice or crystal*.

"I", "C", "c", "N", nMOS and cMOS transistor records are accepted and reformated. The so-called attribute records "A" in the Berkeley format and the "=" records in the MIT format are reformated as comments.

Simfilter requires a **.sim** file, *infile* including any/all extensions for input. Output defaults to stdout.

Additional command line options are:

–n*lambda* Outputs **.sim** file in Berkeley format with technology set to nMOS. Lambda is given in centimicrons and defaults to 200.

–c*lambda* Outputs **.sim** file in Berkeley format with technology set to cMOS-pw. Lambda is given in centimicrons and defaults to 200.

These options would generally be used on output files from *netlist*. Spaces between –n or –c and *lambda* are not accepted.

outfile (optional) An output file name can be specified on the command line.

Note: The default *lambda* values for Berkeley and MIT *Simfilter* makes no assumptions about compatibility of these parameters when reformating Berkeley **.sim** files. This must be done during the extraction of the layout or when running *presim* on the **.sim** file for *rnl*. See the appropriate documentation for details.

SEE ALSO

mextra(1.vlsi), netlist(1.vlsi), presim(1.vlsi), rnl(1.vlsi)
cifplot(1.vlsi),

AUTHOR

Rob Daasch (UW/NW VLSI Consortium, University of Washington)

BUGS

Currently *netlist* only knows about nMOS.

NAME

simtools – Introduction to simulation support tools for VLSI

DESCRIPTION

This document is an introduction to the tools in the UW/NW-VLSI Consortium collection that are used in connection with the simulation of VLSI circuits. These fall into two categories. The first category is the set of simulators and circuit checkers. The second is a set of filters and utilities that are useful for preparing inputs to the simulators and for converting from one file format to another.

We note here that the circuit extractor *mextra* is used for obtaining **.sim** files from layouts. The **.sim** file format is in the Berkeley style (see *simfilter* below).

The simulators and checkers are:

spice2g6 This is the well known device level circuit simulator.

rnl This is an event driven "timing" simulator. It uses logic levels and a simplified circuit model to estimate timing delays through digital circuits. It also has a mode that allows it to be used as a switch (gate) level simulator.

mossim This is a gate level simulator. It uses logic levels and models transistors as perfect switches.

crystal This is a static timing verifier. It uses a simplified circuit model to estimate the worst case delay through a circuit.

erc This is a static electrical rules checker. Performs a set of consistency checks on the circuit.

The filters and utilities that are included in the distribution are:

forfor This is a filter that converts a file using FORTRAN style carriage control to one using ASCII control characters.

simfilter This is a filter that converts Berkeley style **.sim** files such as produced by *mextra* to MIT style **.sim** files used as inputs by programs such as *presim*.

It also can be used to produce Berkeley style **.sim** files from the MIT format allowing circuits described with *netlist* to be analyzed by *spice2g6* or run through the timing verifier *crystal*. This file is acceptable input for *sim2spice* as modified by the UW/NW VLSI Consortium.

netlist This is a program for generating circuit descriptions (net lists). The output is an MIT style **.sim** file.

presim This program converts MIT style **.sim** files into the binary format required by *rnl*. In the process *presim* simplifies the circuit by identifying gates in the circuit.

pspice is a *csh* script that runs *sim2spice* and *spcpp*. In addition to running those programs it concatenates various files so as to create a complete *spice* input deck. This considerably reduces the effort of simulating variations on a particular circuit.

sim2spice This reads a **.sim** (Berkeley style) file containing a description of a circuit. It writes a **.names** file and a **.spice** file. The former contains a translation file from node names in the **.sim** file to the *SPICE* node numbers. The **.spice** file contains a description of the devices in the circuit in a form acceptible to *SPICE*.

spcpp This is a pre-processor to facilitate the writing of *SPICE* input by allowing the user to refer circuit nodes using mnemonic node labels rather than *SPICE* node

numbers. Because *spcpp* can use the .names file produced by *sim2spice* as the translation table from labels to numbers.

spice This is a *csh* script for running *spice2g6*. The usage is:

spice *infile outfile*

The input file *infile* is converted to upper case, and is piped to *spice2g6*. The output of *spice2g6* is piped to *forfor* and is written in file *outfile*.

For more information about any of the above programs, see the appropriate manual entry.

NAME

spcpp – Spice (circuit simulator) input pre-processor

SYNOPSIS

spcpp [**–c**] [**–s** *n*] [**–d** *lr*] [**–t** *tname*] [**–o** *oname*] *iname*

DESCRIPTION

Spcpp is a program that translates bracketed text tokens in an input file into other text strings. It is intended to allow users of *spice* to prepare their simulation input using mnemonic node names rather than the numeric node numbers required by *spice*. The program has two major modes of operation. If the user does not specify a file that contains a translation table, then *spcpp* builds a translation table itself numbering the tokens from zero as it encounters them. Alternatively, the user can specify the name of a file containing a translation table to be used. In particular, the **.names** file created by *sim2spice* is usable as a translation table file.

The options and parameters are:

–c Indicates that the first non-whitespace word of each line of the translation table file should be skipped over. This is useful if your translation table has an asterisk ('*') in column 1 of each line to allow it to be read by *spice* as comments.

–s *n* Indicates that *n* lines at the beginning of the translation table file should be skipped over. If no number is specified then only the first line of the file is skipped.

–d *lr* Redefines the token delimiters to be '*l*' and '*r*' respectively. The default delimiters are '<' and '>'.

–t *tname* Specifies a file that contains a translation table (default is to build a translation table as described above). Each line of this file should have at least two non-whitespace words on it. If the **–c** option is specified then the first word on each line is ignored. The next word is interpreted as a string to be translated and following one is interpreted as the target string into which it is translated. Any subsequent words on the line are ignored. For *spice* input preparation the target string should be a numeral. The **–s** option allows the file to be prefaced by one or more lines that *spcpp* will ignore.

–o *oname* Specifies a file into which the output is to be written. If this option is not used then the output is written to *iroot*.**spcx** where *iroot* is obtained by stripping away any tags from *iname*.

iname Specifies the name of the file to process.

A bracketed token is defined to be a left delimiter character, zero or more spaces, a word (the token) not containing either right or left delimiters, zero or more spaces, and a right delimiter character. Unmatched delimiter characters are not allowed in any context. Bracketed tokens are not allowed to span lines. Tokens and the strings that they translate into are limited to be at most 40 characters each.

Any line that contains no bracketed tokens is simply copied from the input to the output. If a line does contain a bracketed token then the input line is written into the output as *spice* comment line. An output line follows immediately. If the line is valid, then the output line has the untranslated parts immediately below the corresponding parts of the commented input line with the target strings substituted for the bracketed tokens. If an error is detected, then the output line has a caret ('^') immediately below the point at which the first error is detected. An error message line then follows. Since the scanning of the line is abandoned there may be subsequent undetected errors in the remaining part of the line.

Example:

If the following lines are contained in the translation table file:

```
Vdd 1
Input 55
Output 107
foo 23
bar 45
```

then *spcpp* will, upon seeing the lines:

```
.plot trans v(<Input>) v(<Output>), i(<Vdd>)
+ v(<foo>), v(<bar>)
```

will output the lines:

```
* .plot trans v(<Input>) v(<Output>) v(<Vdd>)
  .plot trans v(55) v(107) v(1)
* + v(<foo>), v(<bar>)
  + v(23), v(45)
```

Note that *spcpp* correctly handles *spice* continuation cards.

Note also that the substitution process is not recursive. That is, once a token has been translated, the translated string is not rescanned.

The usefulness of *spcpp* for simulating a circuit extracted from a layout depends upon the user being able to ensure that his mnemonic node labels will be retained through the extraction process. The *mextra* and *sim2spice* manual entries will help with this.

Pspice is a shell script that runs *sim2spice* and *spcpp* and concatenates several files is useful for preparing *spice* inputs from .sim files.

FILES

iname
iroot.spcx
oname
tname

SEE ALSO

mextra(1.vlsi), pspice(1.vlsi), sim2spice(1.vlsi), simtools(1.vlsi), spice(1.vlsi),

SPICE User's Guide, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (*SPICE Version 2G6 User's Guide*, A. Vladimirescu, A.R. Newton, D.O. Pederson, 15 October 1980)

AUTHOR

Robert Fowler (UW/NW VLSI Consortium, University of Washington)

DIAGNOSTICS

The error messages are intended to be self explanitory. If *spcpp* encounters a syntax error on a line then it suspends processing on that line and writes it as a *spice* comment to the output file. It then writes a line containing a caret ("^") under the character at which scanning failed and finally, a line containing an error message. It then goes on to process the remaining lines of the file. If errors have been encountered then at the end of the output file *spcpp* writes messages to the effect that errors have been encountered and exits with status 1. The error

messages written to the output file begin with dollar signs. In addition, some number of messages are directed towards the standard error output.

BUGS

The target strings are not checked to see whether they are valid numerals or not. This can be regarded as either a bug or a feature.

The target string must fit into the space from the left to right token delimiter inclusive. This is normally not a problem since most node numbers will be small integers and the available space will be at least three characters. This was done so that the input lines and the translated outputs would line up vertically.

NAME

spice – circuit simulator

SYNOPSIS

spice *infile outfile*

DESCRIPTION

Spice reads a circuit description from *infile*. Output is written to *outfile*. and error messages to standard error.

Spice is a general-purpose circuit simulation program for nonlinear DC, nonlinear transient, and linear AC analyses. Circuits may contain resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four types of dependent sources, transmission lines, and the four most common semiconductor devices: diodes, BJTs, JFETs, and MOSFETs.

Spice has built-in models for the semiconductor devices, and the user need specify only the pertinent model parameter values. The model for the BJT is based on the integral charge model of Gummel and Poon; however, if the Gummel-Poon parameters are not specified, the model reduces to the simpler Ebers-Moll model. In either case, charge storage effects, ohmic resistances, and a current-dependent output conductance may be included. The diode model can be used for either junction diodes or Schottky barrier diodes. The JFET model is based on the FET model of Shichman and Hodges. Three MOSFET models are implemented; MOS1 is described by a square-law I-V characteristic, MOS2 is an analytical model while MOS3 is a semi-empirical model. Both MOS2 and MOS3 include second-order effects such as channel length modulation, subthreshold conduction, scattering limited velocity saturation, small size effects and charge-controlled capacitances.

To build a *spice* input file for your circuit from *mextra* output run *sim2spice* or *pspice*.

Note: In the UW/NW VLSI Consortium distribution, *spice is a csh* script that translates *infile* to upper casepipes that to the current version of the *spice* simulator, and then pipes the output through *forfor*, a filter that converts FORTRAN style carriage control to UNIX compatible codes.

FILES

/usr/local/spice circuit simulation program

SEE ALSO

mextra(1.vlsi)
sim2spice(1.vlsi), pspice(1.vlsi), spcpp(1.vlsi)

SPICE User's Guide, **VLSI Design Tools Reference Manual**, UW/NW VLSI Consortium, University of Washington, (*SPICE Version 2G6 User's Guide*, A. Vladimirescu, A.R. Newton, D.O. Pederson, 15 October 1980).

Program Reference for Spice2, E. Cohen, ERL Memo. ERL-M592, Electronics Research Laboratory, University of California, Berkeley, June 1976.

SPICE2: A Computer Program to Simulate Semiconductor Circuits, L.W. Nagel, ERL Memo. ERL-M520, Electronics Research Laboratory, University of California, Berkeley, May 1975.

The Simulation of MOS Integrated Circuit Using SPICE2 A. Vladimirescu and Sally Liu, UCB/ERL M80/7, University of California, Berkeley, February 1980.

AUTHOR

(UCB)

BUGS

MOSFET Model, Level=2 does not work, due to a charge conservation problem (it grows).

NAME

tech – VLSI technology database files

SYNOPSIS

technology.tec
technology.cmp

DESCRIPTION

Files of this form are used by the VLSI plotting programs *dbdi* and *penplot*. *Technology* would most commonly be **nmos** or **cmos**.

The *technology*.tec file contains the name of the colormap (*technology*.cmp) to be used and a table of parameters for each layer, as shown in the following listing of the default *nmos.tec* file.

```
nmos
NM      f       4       4       3       metal
NP      f       1       1       2       polysilicon
ND      f       2       2       2       diffusion
NI      f       8       8       0       implant
NC      x       35      255     2       contact
NB      o       32      255     0       buried
NH      o       33      255     0       himplant
NX      o       34      255     0       extra
NG      o       255     255     0       glass
NM2     f       33      255     0       metal2
NC2     o       34      255     0       contact2
LL      s       35      255     0       label
XP      o       255     255     0       pads
TX      s       35      255     0       text
BG      s       0       255     0       background
CR      o       64      64      0       cursor
```

Here, *nmos* refers to *nmos.cmp*, which is the colormap to be used. The columns are defined as follows:

1) CIF layer name;
2) fill type (f=filled, x=X'ed, o=outlined);
3) color map index (see below);
4) write mask (255=opaque, others control amount of transparency);
5) minimum wire width in lambda;
6) layer name.

The *technology*.cmp file contains the color map used by the plotting programs. There are 256 entries (0–255) and all must be specified. Below is a portion of the default *nmos.cmp* file:

```
200     200     200     0       light background
220     0       120     1       red
70      250     70      2       green
160     160     0       3
0       0       200     4       blue
115     0       155     5
0       170     140     6
115     115     125     7
250     250     0       8       yellow
200     100     60      9
```

The columns are defined as follows:

1) RED value (8 bits);
2) GREEN value (8 bits);
3) BLUE value (8 bits);
4) color map index (as referenced in *technology*.tec);
5) color name.

The user's search path is searched for these files, so the user may create his own technology files in a directory spicified in his search path. The default versions of these files are **/usr/vlsibin/***technology***.tec** and **/usr/vlsibin/***technology***.cmp**, therefore, the user should have the directory name **/usr/vlsibin** somewhere in his search path definition for them to be accessible.

SEE ALSO

dbdi(1.vlsi), penplot(1.vlsi)

PLAP User's Guide

UW/NW VLSI Consortium
University of Washington
Seattle, WA 98195

This manual corresponds to PLAP version 2.0.

1. Introduction

LAP is a package of artwork generation routines, originally written at Caltech by Bart Locanthi. It generates integrated circuit artwork data files in the *C*altech *I*ntermediate *F*orm (*CIF*) in response to procedure calls executed within a program written by the VLSI designer. Boeing **PLAP** is a complete re-design of the original software. It is written in Pascal rather than SIMULA, has many additions to improve the user interface, and removes former restrictions of maximum chip complexity.

PLAP allows either the entire design, a group of cells, or a single cell to be contained in a program. Thus, if only one symbol needs to be altered, only the program containing that symbol needs to be re-compiled and re-executed. The result is a minimization of the compilation and execution times required to design an integrated circuit, assuming a careful decomposition of the design.

In order to communicate symbol information between separate **PLAP** programs, each program creates two files for each symbol defined in the **PLAP** code. The first is the *.sym* file, which contains modified *CIF* code with symbol numbers replaced by symbol names. The second is the *.att* file, which consists of bounding box coordinates, a list of the symbols called, and a list of user-definable attributes. These attributes are (name,value) pairs where the attribute name is associated with an arbitrary real number. They may be used at the designer's discretion and routines are provided for setting and retrieving attributes. Both symbol files are read-only protected to prevent any manual editing of their contents. To produce a "standard" *CIF* file, the DB2CIF conversion program must be run. This collects all the *.sym* files required to completely describe the given symbol.

In order to facilitate the use of a large data base of symbol files, **PLAP** makes use of a symbol data base organized as cell libraries. There may be up to *three* libraries declared, but there must be *at least one* specified. The current working directory should never be used as one of the libraries to avoid a clutter of **PLAP** programs and symbol files. The symbol files will be created in the last library declared in the **PLAP** program.

2. How to Use Boeing PLAP v.2

2.1. Program Shell

A small shell file is provided which sets up the proper global variable environment for linking with the PLAP modules. User-defined variables and procedures must be inserted in appropriate places *following* the PLAP system declarations. This template file */usr/lib/uw-vlsi/pshell.p* is shown below.

```
{ ************************************************************************}
{* basic shell for VLSI artwork design language                          *}
{ ************************************************************************}

program shell

#include '/usr/include/plap/prog.h'              { program parameters }

{ declarations section }

const
#include '/usr/include/plap/gconst.h'             { global constants }
{** insert user-defined CONSTants here **}

type
#include '/usr/include/plap/gtype.h'               { global types }
{** insert user-defined TYPEs here **}

var
#include '/usr/include/plap/gvar.h'               { global variables }
{** insert user-defined VARiables here **}

{ procedure declarations }

#include '/usr/include/plap/primitive.h'          { LAP primitives }
#include '/usr/include/plap/macros.h'              { LAP macros }
#include '/usr/include/plap/nodes.h'           { LAP utilities source }
#include '/usr/include/plap/error.i'              { LAP internals }
{** insert user-defined PROCEDUREs here **}

begin  {** main program **}

{ declare up to three libraries }
  initialize(technology,'logfileName');
  library(librarydirectory);

{ ************************************************************************}
{* insert VLSI design definitions here                                   *}
{ ************************************************************************}

{ ************************************************************************}
{* end of user VLSI design definitions                                   *}
{ ************************************************************************}
```

```
    wrapup;

1:                                  { fatal error exit point }

end. {** main program **}
```

The **#include** statements cause external text files to be inserted at the locations which contain the actual declarations. These files must be the same files used to compile the **PLAP** system modules. Users should verify that the pathnames are the ones actually used on their system.

The **initialize** and **library** calls must precede any other main program statements, and the **wrapup** call must be the last statement in the program.

Note: On UNIX, Pascal keywords must be in lower case and the case of variables *is* significant. Therefore, users should be aware of the possibility of this type of error.

2.2. Compile, Link, and Run

The **PLAP** design file is a Pascal program which must be compiled, linked with external **PLAP** modules, and executed. A command file is provided which performs all of the above for the user. Simply type:

% plap [-clf] *basename*

The file extension is not necessary in *basename*.

The **PLAP** program will execute, listing symbol names on the terminal as they are defined and displaying any error messages that might occur. All information displayed on the terminal is also echoed into the log file whose name was entered in the **initialize** statement. In addition, **PLAP**'s interpretation of any PLA code files will be echoed into the log file for confirmation by the user.

After processing the user's program, **PLAP** will give the prompt:

menu: w=write, s=show, q=quit, a=abort?

Typing **s**<*return*> will cause all the symbols in the symbol tree to be listed, along with the bounding box of each symbol. Typing **q**<*return*> will terminate the program if a **write** has already been performed. Typing **a**<*return*> will terminate the program without producing any *.sym* or *.att* files. Typing **w**<*return*> will cause the *.sym* and *.att* files to be written onto the disk in the user's account. They will be written into the last library declared by a **library** call in the **PLAP** design file.

The **-clf** option runs DB2CIF on *basename*.**sym** to produce *basename*.**clf**. This will only work, of course, if a symbol is defined with the name of *basename*.

2.3. Units and Parameter Types

All dimensions are in lambda (Mead & Conway), that is, one half of a minimum polywire width. The absolute value of lambda is set by a **PLAP** constant; the default is 2.5 microns. It can be changed by the **lambda** procedure call.

Parameters passed to **PLAP** commands must adhere to certain Pascal type-checking rules. For instance, all dimensions or coordinates must be of type *real*. Other parameters are of types such as *integer* or *string*, as described for each command in Sections 3 and 4 of this manual.

Parameters can be literal constants (numbers) as well as variables. Under UNIX, real numbers need a decimal point and at least one digit preceding and following the decimal point. (The UNIX Pascal compiler may create illegal binary code if real constants are not followed by a decimal point and one digit.)

---

†UNIX is a Trademark of Bell Laboratories.

A string can be either a literal enclosed in single quotes, or a variable. Strings can be any length, within the following constraints:

- String parameters specifying symbol names (in **define** and **draw** commands) are truncated to 9 characters. Other text string parameters may be up to 80 characters in length.
- Strings do not need to be padded with blanks to a fixed length, and trailing blanks are ignored.
- String parameters must be at least 2 characters, however the second character may be a blank, which would then be ignored.

3. Detailed Discussion of Commands

3.1. Artwork Primitives

layer(*layerName***);**

This command sets the mask layer for succeeding art work. The following tables relate **PLAP** layer names to the *CIF* layers for *nMOS* and I^2L technologies:

nMOS Parameters:

layerName	CIF layer
polysilicon, poly or red	NP
metal or blue	NM
metal2	NM2
diffusion or green	ND
implant or yellow	NI
himplant or capacitor	NH
contact, contax, cuts, cut, black	NC
contact2, contax2, cuts2, cut2, via	NC2
glass	NG
buried	NB
extra	NX

I^2L Parameters:

layername	CIF layer
metal or blue	IM
metal2	IM2
emitter or green	IE
base or red	IB
sinker or yellow	IS
glass or brown	IG
contact, contax, cuts, cut, black	IC
contact2, contax2, cuts2, cut2, via	C2
buried	IBL
isolation	II
resistor	IR

Examples:

```
layer(polysilicon);
layer(blue);
layer(sinker);
```

box(*x1*,*y1*,*x2*,*y2*);
lbox(*layerName*,*x1*,*y1*,*x2*,*y2*);

layerName Layer name as described for the **layer** command.

x1,*x2*,*y1*,*y2* Coordinates in lambda units (real);

Both of these procedures draw a box with lower left corner at (*x1*,*y1*) and upper right corner at (*x2*,*y2*). Box edges are always aligned with respective *x*- and *y*-axes.

The **lbox** procedure causes a permanent change in the mask layer before the box is generated.

Although a warning message is produced if a box has zero width or length, it is still output to the *CIF* file.

Examples:

```
box(-2.9,7.0,9.5,26.0);
lbox(metal,13.0,0.5,19.0,9.5);
```

wire(*width*,*xStart*,*yStart*); *wirePath*
lwire(*layerName*,*width*,*xStart*,*yStart*); *wirePath*

layerName Layer name as described for the **layer** command;

width Width in lambda units (integer);

xStart,*yStart* Coordinates in lambda units (real).

wirePath An arbitrary sequence of wire path primitive commands (Section 3.2). A wire path is automatically terminated by any **PLAP** command that is not one of these primitives.

Both of these commands draw a wire of width=*width* starting at (*xStart*,*yStart*) and continuing as specified in the *wirePath*. The **wire** procedure draws the wire in the same layer as last drawing command. The **lwire** procedure changes the layer to *layerName* before drawing the wire.

Examples:

```
wire(2.0,-7.0,2.5);
    x(23.0);y(7.0);xy(35.0,17.5);z(blue);dx(-4.0);
lwire(blue,3.0,9.0,6.0);
    dxy(12.0,9.0);w(10.0);x(21.0);y(17.5);
```

polygon(*xStart*,*yStart*); *polyPath* {**polyclose**};

xStart,*yStart* Coordinates in lambda units (real);

polyPath An arbitrary sequence of wire path statements, *excluding* the **z** and **w** commands (Section 3.2).

polyclose An optional command to close the polygon (generate a final vertex equal to the first vertex). If a polygon is not closed or has less than three sides, an error message will be generated.

This procedure generates an arbitrary polygon. *It should be used only if absolutely necessary and when the user is willing to take the risk that downstream analysis programs may not accept polygons.* For instance, *FASTDRC*, *ALLDRC* and the *MEXTRA* extractor ignore all polygons.

If a polygon statement is used anywhere in the program, a warning message will be given at the end of program execution.

Examples:

```
polygon(1.0,3.5);
    x(5.0);dy(2.5);x(-4.0);y(-2.0);x(1.0);y(3.5);
polygon(1.0,3.5);
    xy(5.0,6.0);x(-4.0);y(-3.5);x(1.0);polyclose;
```

3.2. Path Primitives

xy(*xNext,yNext***);**
dxy(*xDelta,yDelta***);**
x(*xNext***);**
y(*yNext***);**
dx(*xDelta***);**
dy(*yDelta***);**

xNext,yNext Coordinates in lambda units (real);

xDelta,yDelta Lengths in lambda units (real);

Procedures **x**, **y** and **xy** create a wire from the current position to the point with the specified new *x*- and/or *y*-coordinate.

Procedures **dx**, **dy** and **dxy** create a wire from the current position to the point with the specified change to the *x*- and/or *y*-coordinate.

z(*newLayerName***);**

newLayerName Layer name as described for the **layer** command;

The **z** procedure changes the current layer to be *newLayerName*. Subsequent commands in the wire path will create wires on this layer, until changed again. This command is only appropriate in a *wirePath*, not in a *polyPath*.

w(*newWidth***);**

newWidth Width in lambda units (integer);

The **w** procedure changes the current wire width to be *newWidth*. Subsequent commands in the wire path will create wires of this width, until changed again. This command is only appropriate in a *wirePath*, not in a *polyPath*.

3.3. Symbol Definition and Call Statements

define('*symbolName***');**
endef;

symbolName A string constant or variable (see Section 2.3 for a description of acceptable strings).

This pair of statements is used to bracket a symbol definition. A symbol must be defined before it can be used. All artwork statements must be contained within symbol definitions, otherwise a warning message will be produced.

The symbol name must be unique and must not correspond to any of the following internal **PLAP** symbol names:

rb	PLAconnect	PLAinput	PLAtee	pulldownN
gb	PLApullup	PLABinput	PLAspace	pulldownS
greast	PLABpullup	PLAoutput	pulldownE	

PLAcell PLAground PLABoutput pulldownW

Lower and uppercase are significant.

Symbol definitions may not be nested. This restriction is due to *CIF* not allowing nested definitions.

Examples:

```
define('latch');
  .
  .
enddef;

define('REGISTER_16');
  .
  .
enddef;
```

draw('*symbolName*',*x*,*y*);
drawmx('*symbolName*',*x*,*y*);
drawmy('*symbolName*',*x*,*y*);
drawrot('*symbolName*',*x*,*y*,*vx*,*vy*);

symbolName Symbol name as described for the **define** command.

x,y Coordinates in lambda units (real);

vx,vy Coordinates (integer) of the vector endpoint starting at the origin;

These procedures cause a symbol with name *symbolName* to be drawn with its origin at (*x*,*y*). The symbol must be defined before it can be drawn.

drawmx first mirrors the symbol in the *x*-dimension (exchanges -*x* for *x*) and **drawmy** first mirrors the symbol in the *y*-dimension (exchanges -*y* for *y*). **drawrot** first rotates the symbol using the rotation vector(*vx*,*vy*). The symbol is rotated through the same angle that the rotation vector is rotated from the *x*-axis. For instance, the following rotation vectors correspond to the these rotations:

(1, 0)	no rotation
(0, 1)	90 degrees
(-1, 0)	180 degrees
(0,-1)	270 degrees
(1, 1)	45 degrees

The rotation vector can be of any magnitude except zero. *The user should be warned that non-orthogonal rotations may cause difficulty in downstream analysis programs and thus should be avoided.*

Examples:

```
draw('latch',23.0,37.5);
drawmx(plal".planame,0.0,-23.0);
drawrot('datapaths',7.0,13.0,0,1);
```

3.4. Functions and Annotation Statements

lastX;
lastY;

These functions return the most recent *x* or *y* coordinate (real) used in a series of wire path statements. **box** or **draw** commands do not set these values.

Examples

```
wire(2.0,7.0,13.0);x(11.0);y(21.0);z(blue);
wire(2.0,lastX,lastY);x(-2.0);dy(-2.0);
```

plottext('*textString*',*x*,*y*,*centered*);
nodelabel('*textString*',*x*,*y*,*layerName*);

textString A string constant or variable, up to 80 but at least 2 characters in length. Trailing blanks are ignored.

x,y Coordinates in lambda units (real);

centered Boolean flag;

layerName Layer name as described for the **layer** command.

These statements serve to annotate the Versatec plots created by the Berkeley *CIFPLOT* program, and to name nodes for the *MEXTRA* extractor.

plottext plots a string of text at (*x*,*y*) on the Versatec plot. If *centered="true", the text is centered on* (*x*,*y*); otherwise (*x*,*y*) is the lower left corner of the text. The text is always horizontal. This is useful for labeling the plot with date and version of the design.

nodelabel plots a text string at (*x*,*y*) on the Versatec plot and attaches a node name to any feature intersecting (*x*,*y*) on layer *layerName*. For *CIFPLOT*, (*x*,*y*) is the lower left corner of the text and *layerName* is significant only in that if that layer is excluded from the plot, the string will also be excluded. Node labels are required for the *MEXTRA* extractor, *ERC* and *MOSSIM* simulator. (See the user's guides for those programs.)

Examples:

```
plottext('version8/23/81',100.0,239.0,false);
nodelabel('muxinput',10.0,22.5,metal);
```

3.5. Boolean Switch Variables

checkswitch := *boolean*;
manhattan := *boolean*;
halfcheck := *boolean*;

These are global Boolean variable switches that the user may override.

checkswitch False by default, but if set to true, causes the program to check for minimum wire widths in all wires.

manhattan True by default and causes the program to check that all geometry in the user's design contains only 90 degree angles.

halfcheck True by default and causes the program to check that all figures in the user's design fall only on 1/2 lambda boundaries.

Any of the above switches may be set by assignment statements placed after the **initialize** statement and before any user VLSI design functions.

Examples:

```
checkswitch := true;
manhattan   := false;
halfcheck   := false;
```

3.6. Macros

downroute(*nLines,xStartArray,yStart,xEndArray,yEnd,width,spacing***);**

nLines	Number of wires to route (integer), where *nLines* ≤ 50;
yStart,yEnd	*y*-coordinate in lambda units (real);
xStartArray,xEndArray	Array of *nLines* *x*-coordinates in lambda units (real). This is a predefined type called routarray;
width	Width in lambda units (integer);
spacing	Minimum wire spacing in lambda units (integer).

This procedure performs a simple river-routing in the current mask layer, with right-angled corners. It only works in the vertical direction, preferably top-down (bottom-up may still have some bugs). It connects an array of wires, all starting at *y=yStart*, with initial *x*-coordinates in *xStartArray*, to points at *y=yEnd* with final *x*-coordinates in *xEndArray*. All *x*-coordinates in the two arrays must be in ascending order, and *no crossings are allowed*. The wires will be drawn with width and spacing as specified by the last two parameters.

Example:

In the declarations section:

```
var startxs,endxs: routarray;
```

In the execution section:

```
layer(poly);
for i := 1 to 23 do
  begin
  startxs[i] := 8*i;
  endxs[i] := 102 + 9*i;
  end;
downroute(23,startxs,0.0,endxs,-120.0,2,2);
```

3.7. Symbol Access Statements

library('*libraryName***');**

libraryName A full Unix path name.

The library procedure is used to establish a search path for **PLAP** to follow when looking for a particular symbol. In searching for a symbol, **PLAP** will search the libraries in the reverse order in which they were declared and it stops as soon as it finds the first occurrence of the symbol. The user *must* declare his own working library and he may *optionally* declare group project and system libraries. The libraries must be declared before any symbols are defined. A separate **library** command must be used for each of up to *three* library declarations.

Example:

```
begin
  initialize(...);
  library('/u1/larry/lib');
    .
    .
  wrapup;
end.
```

withsym(*'symbolName'*);

symbolName Symbol name as described for the **define** command.

This procedure is used in conjunction with the **fa** function and **setatt** procedure to specify the symbol currently being accessed.

Example:

```
var x : real;

begin
  initialize(...);
  withsym('cell');
  x := fa('vddx');
  setatt('vddy',0.0);
    .
    .
  wrapup;
end.
```

In this example, the symbol being accessed is "cell". The call to **fa** will set variable x to be equal to the "vddx" value from "cell" and the call to **setatt** will set the "vddy" attribute for "cell1" to be zero.

fa(*'attributeString'*);

attributeString may be one of the following:

1. A string containing only the attribute name, in which case the attribute associated with the symbol name used in the last call to the **withsym** procedure.
2. A string of the form *symbolName.attributeName*, in which case the attribute is assumed to be associated with the symbol name specified in the string.

The fetch attribute function returns the value of an attribute of a symbol as a real number. The attribute must have been set beforehand by the **setatt** procedure.

Example:

```
var x1,x2 : real;

begin
  initialize(...);
    .
    .
  withsym('cell1');
  x1 := fa('gndx');
  x2 := fa('cell2.gndx');
    .
  wrapup;
end.
```

In the example, x1 will be set to the value of "gndx" from "cell1" and x2 will be set to the value of "gndx" from "cell2".

setatt(*'attributeString',value*);

attributeString An attribute name as described for the **fa** function.

value The value (real) to be assigned to the attribute name.

This procedure sets the attribute name for a symbol to have the corresponding value.

Example:

```
begin
  initialize(...);
    .
    .
  withsym('cell2');
  setatt('cell1.gndx',15.5);
  setatt('gndx',10.5);
    .
  wrapup;
end.
```

The first call to **setatt** will set the attribute "gndx" of the symbol "cell1" to have the value 15.5 and the second call will set the attribute "gndx" of the symbol "cell2" to be 10.5.

3.8. Lambda Definition

lambda(*microns*);

microns Number of microns per lambda unit (real).

The **lambda** command sets the global definition of lambda for the design. It must be used in each **PLAP** program of the design and must be set before the first define in each program. If this command is not used, **PLAP** assumes lambda=2.5 microns. If lambda is changed in the course of a design, it is important that all previously defined symbols are re-**PLAP**ed with the new lambda value.

Example:

```
lambda (2.0);
```

This statement would set the scale on each *CIF* symbol definition to be 200/20.

4. nMOS Macros

4.1. Basic Macros

rb(*x*,*y*);
gb(*x*,*y*);
be(*x*,*y*);
bn(*x*,*y*);
bs(*x*,*y*);
bw(*x*,*y*);

x,*y* Coordinates in lambda units (real).

These macro calls generate symbol calls for various contacts. **rb** and **gb** are red-blue (poly-metal) and green-blue (diffusion-metal) contacts, respectively, centered at (*x*,*y*). The remaining four calls generate butting contact symbol calls in various orientations. The butting contact origin is one lambda from the diffusion end, and is placed at (*x*,*y*). **be** places the poly end eastward, **bn** northward, etc.

Examples:

```
rb(26.0,39.5);
bn(-7.5,2.0);
```

pullup(*x,y,polyLength,direction***);** ***drainPath***

x,y Coordinates in lambda units (real);

polyLength Length of poly gate in lambda units (real);

direction Orientation of contact and gate: *east*, *north*, *west* or *south*;

drainPath A sequence of wire path primitive commands.

This macro call generates a depletion-mode pullup with butting contact at (*x,y*) with a 6-lambda wide poly gate of length *polyLength*. The butting contact and poly gate are oriented in the direction specified by *direction*. A 2-lambda wide diffusion wire is generated with its centerline running from (*x,y*) to 2-lambda beyond the end of the poly gate. *drainPath* is a set of wire path statements that continue the drain wire from this point.

Examples:

```
pullup(16.0,43.0,8.0,north);dy(5.0);x(21.0);
pullup(0.5,-3.0,10.0,south);dy(-4.0);x(blue)
```

alpha(*'textString',x,y,scale***);**

textString A string constant or variable up to 80 characters long;

x,y Coordinates in lambda units (real);

scale Text scaling factor (real).

This macro call uses wires to generate letters in the current mask layer. (*x,y*) is the lower left corner of the string, and the letters are created with

height = 8**scale* (lambda),
width = 5**scale* (lambda),
spacing = 2**scale* (lambda),
thickness = *scale* (lambda).

The following characters are supported:

A..Z, 0..9,
- . , ? / { } + =

Lower case is converted to upper case, but any other character will result in a blank space.

The **alpha** command is very expensive in computer time and CIF file space. ***It should not be used for extensive plot annotation.*** For plot annotation, use **plottext** or **nodelabel**.

Examples:

```
alpha('8X8MultiplierVersion2/17/81',103.0,79.5,2.0);
alpha('date-out',5.0,27.0,3.0);
```

4.2. Pad Macros

inpad(*padSpace,width***);**
outpad(*padSpace,width***);**
padtristate(*spacing***);**

These macro calls generate artwork for various pads. The spacing parameters extend the power and ground buses and the width parameter fattens the *Vdd* line beyond its basic 4-lambda width.

Default values (real) are assigned to the parameters when they are negative, with default assignments as follows:

padSpace = 80.0
width = 0.0
spacing = 4.0

Each of these procedures must be called within a symbol definition for the desired pad. Then **draw** statements may be used to place the pads in the design.

Examples:

```
define('input-pad');
  inpad(80,8);
  enddef;

define('output-pad');
  outpad(100,-1);
  enddef;

define('tristate-pad');
  padtristate(8);
  enddef;

drawrot('output-pad',100,273,0,1);

drawmy('tristate-pad',-74,91);

for i := 1 to 8 do
  draw('input-pad',80*i,173);
```

4.3. PLA's

4.3.1. PLA Command Descriptions

newpla(*nInputs*,*nMinterms*,*nOutputs*,*nFeedbacks*,'*codeFileName*',*gutSwitch*,*clockSwitch*);

nInputs	Total number of PLA inputs (integer), including feedbacks;
nMinterms	Number of minterms (integer), these are rows in the programming matrix, (must be an even number);
nOutputs	Total number of outputs (integer), including feedbacks, (must be an even number);
nFeedbacks	Number of feedback terms (integer), these are outputs which directly connect to inputs of the same PLA;
codeFileName	A string containing the name of the code file for programming the PLA. It *must not* include directory information or extension. **PLAP** assumes an extension of *.cod*, and assumes that the file is located in the user's local directory. This name will subsequently be used as the PLA name;
gutSwitch	A Pascal enumerated type and is equal to either *includeguts* or *noguts*. If it is equal to *noguts* only the border of the PLA is generated;
clockSwitch	An enumerated type, equal to either *clocked* or *noclock*. It controls whether poly clock gates are included on the PLA input and output circuits.

This macro function-call initializes a PLA and creates an attribute record containing input and output and other coordinates that can be later accessed by the user via a pointer variable. **newpla** does not create the artwork for a PLA, this must be done with the **draw** command. The PLA is similar to the one written by Dave Johannsen for the 1979 Carver Mead U.W. VLSI Design course, but has been improved to eliminate extraneous internal contacts.

Example:

```
var plal : plapointer;

begin
  initialize(...);
  halfcheck := false;
  plal := newpla(4,16,4,0,'adder',includeguts,clocked);
    .
    .
  wrapup;
end.
```

An extensive example is given in Section 6.

Note: In order to minimize space requirements, the PLA artwork procedures employ rectangles which are not necessarily on half lambda boundaries. Consequently the variable *halfcheck* must be set to false before **newpla** is called.

getpla(*PLAname***);**

PLAname The name of the PLA (the same name used for the code file in the call to the **newpla** function).

The function **getpla** is used to access information about a PLA which was defined in a previous *PLAP* program. It returns a pointer to a PLA record of the same type as the **newpla** function.

Example:

```
var plal : plapointer;

begin
  initialize(...);
  plal := getpla('adder');
    .
    .
  wrapup;
end.
```

The user may access any information about the PLA which is normally available in a call to **newpla**.

4.3.2. Connecting Wires to the PLA

The **newpla** command is a function call that returns a pointer variable, pointing to a Pascal record of the following type:

```
record {PLA attribute record type}
  inputs, minterms, outputs, feedbacks   : integer;
  planame   : string;
  llx, lly, urx, ury   : real;
  inputy, outputy   : real;
  plaheight, plawidth   : real;
  vddx, vddy, gndx, gndy   : real;
  inclockx, inclocky, outclockx, outclocky   : real;
  inputx   : array [1..maxplainputs] of real;
  outputx   : array [1..maxplaoutputs] of real;
  end;
```

To access any data in this record, use the Pascal pointer notation as shown in the example below:

```
var plal : plapointer;

halfcheck := false;
plal := newpla(12,16,8,3,'cntrl',includeguts,clocked);
draw(plal^.planame,0,0);
lwire(green,2,0,-20);x(plal^.inputx[1]);y(plal^.inputy);
lwire(red,2,0,-10);x(plal^.inclockx);y(plal^.inclocky);
```

The *var* statement declares *plal* as a pointer to a PLA record. This statement must be placed in the initial variable declarations, as are all user variable declarations. The next occurence of *plal* accesses the PLA name, the third accesses the *x* coordinate of the first input, the fourth accesses the *y* coordinate of the inputs, and fifth and sixth access the *x* and *y* coordinates of the input clock.

Vdd connects to the left edge of the PLA, and *ground* connects to the right edge.

4.3.3. PLA Code File Format

The PLA is built with inputs on the bottom left, numbered left-to-right, and outputs on the bottom right, also numbered left-to-right, the left-most input and output being number 1. Feedbacks connect the last input with the first output, the second-to-last input to the second output, etc. Currently PLAP limits the maximum number of inputs or outputs to 75. No limit is placed on the number of minterms.

The format of the code file is as follows. There is a single line for each minterm of the PLA. On each line, the coding of the AND plane is on the left and the OR plane is on the right, separated by a single blank space. Columns in the AND plane correspond to inputs with number 1 on the left, and columns in the OR plane correspond to outputs, again with number 1 on the left. Coding for the AND plane uses characters from the set {X,0,1}, where "X" may be upper or lower case, and where any mnemonic character except "X" may be substituted for a "1".

Coding for the OR plane uses characters from the set {0,1}, with the exception that any mnemonic character except "X" may be substituted for a "1".

If PLAP encounters nonstandard characters in the AND or OR plane, a warning message is generated.

Comments can be inserted in the code file by starting a line with a "!". Blank lines are ignored.

Example:

The code file line:

```
0X1 011100
```

will drive outputs 2, 3 and 4 high when input 1 is low and input 3 is high.

5. Command Summary

5.1. Artwork Primitives

layer(*layerName***);**
box(*x1,y1,x2,y2***);**
lbox(*layerName,x1,y1,x2,y2***);**
wire(*width,xStart,yStart***);** *wirePath*
lwire(*layerName,width,xStart,yStart***);** *wirePath*
polygon(*xStart,yStart***);** *polyPath* {*polyClose*;}

5.2. Path Primitives

xy(*xNext,yNext***);**
dxy(*xDelta,yDelta***);**
x(*xNext***);**
y(*yNext***);**
dx(*xDelta***);**
dy(*yDelta***);**

z(*newLayerName***);**
w(*newWidth***);**

5.3. Symbol Definition and Call Statements

define('*symbolName***');**
endef;
draw('*symbolName***',***x,y***);**
drawmx('*symbolName***',***x,y***);**
drawmy('*symbolName***',***x,y***);**
drawrot('*symbolName***',***x,y,vx,vy***);**

5.4. Functions and Annotation Statements

lastX;
lastY;
plottext('*textString***',***x,y,centered***);**
nodelabel('*textString***',***x,y,layerName***);**

5.5. Boolean Switch Variables

checkswitch
manhattan
halfcheck

5.6. Macros

downroute(*nLines,xStartArray,yStart,xEndArray,yEnd,*
*width,spacing***);**

5.7. Symbol Access Statements

library(*libraryName***);**
withsym(*symbolName***);**
fa(*attributeString***);**
setatt(*attributeString,value***);**

5.8. Lambda Definition

lambda(*microns*);

5.9. nMOS Macros

rb(*x*,*y*);
gb(*x*,*y*);
be(*x*,*y*);
bn(*x*,*y*);
bs(*x*,*y*);
bw(*x*,*y*);
pullup(*x*,*y*,*polyLength*,*direction*); ***drainPath***
alpha('*textString*',*x*,*y*,*scale*);

inpad(*padSpace*);
outpad(*padSpace*);
padtristate(*spacing*);

newpla(*nInputs*,*nMinterms*,*nOutputs*,*nFeedbacks*,
'*codeFileName*',*gutSwitch*,*clockSwitch*);
getpla(*PLAName*);

6. A Complete PLA Example

The following three sections describe in detail the definition and use of a PLA. This example is a simple 2-bit adder. It is implemented by a clocked PLA with 4 inputs, 16 minterms, 4 outputs and 0 feedbacks. The left-most 2 inputs and the right-most 2 inputs provide the 2-bit *addends* and the 4 outputs provides the 4-bit *sum*

6.1. PLA Code File

The following PLA code is contained in a file named *test.cod*. The number of inputs (4) and outputs (4) are evident in the number of columns on the left and right, respectively. The number of minterms (16) are evident in the number of rows.

```
0000 0000
0001 0001
0010 0010
0011 0011
0100 0001
0101 0010
0110 0011
0111 0100
1000 0010
1001 0011
1010 0100
1011 0101
1100 0011
1101 0100
1110 0101
1111 0110
```

6.2. PLA Definition

The following brief **PLAP** program defines the PLA but does not place it in any design. Although the basic template has been used for this example, the comments have been deleted for clarity. The user inserted text is "in boldface."

```
program pshell
   #include '/usr/include/plap/prog.h'

{ declarations section }

const
   #include '/usr/include/plap/gconst.h'
type
   #include '/usr/include/plap/gtype.h'
var
   #include '/usr/include/plap/gvar.h'
   adder : plapointer;

{ procedure declarations }

#include '/usr/include/plap/primitive.h'
#include '/usr/include/plap/macros.h'
#include '/usr/include/plap/nodes.h'
#include '/usr/include/plap/error.i'

begin  {** main program **}

  initialize(nmos, 'adder');
  library('/u1/yanagida/pla_example/lib/');

  halfcheck := false;
  adder := newpla(4,16,4,0,'adder',includeguts,clocked);

  wrapup;

end.  (** main program **)
```

Upon the execution of this program, the symbol named *adder* will have been defined and saved in the library */u1/yanagida/pla_example/lib/*.

6.3. PLA Instantiation

The following brief **PLAP** program uses the PLA defined in the previous sections **PLAP** program. User inserted text is "in boldface."

```
program pshell
   #include '/usr/include/plap/prog.h'

{ declarations section }

const
   #include '/usr/include/plap/gconst.h'
type
   #include '/usr/include/plap/gtype.h'
var
   #include '/usr/include/plap/gvar.h'
   adder : plapointer;

{ procedure declarations }

#include '/usr/include/plap/primitive.h'
#include '/usr/include/plap/macros.h'
#include '/usr/include/plap/nodes.h'
#include '/usr/include/plap/error.i'

begin    (** main program **)

   initialize(nmos, 'example');
   library('/u1/yanagida/pla_example/lib/');

   adder := getpla('adder');

   define('example');
      draw(adder^.planame, 0, 0);

      nodelabel('input1',
            adder^.inputx[1], adder^.inputy,
            green);
      nodelabel('input2',
            adder^.inputx[2], adder^.inputy,
            green);
      nodelabel('input3',
            adder^.inputx[3], adder^.inputy,
            green);
      nodelabel('input4',
            adder^.inputx[4], adder^.inputy,
            green);
```

```
        nodelabel('output1',
                adder^.outputx[1], adder^.outputy,
                green);
        nodelabel('output2',
                adder^.outputx[2], adder^.outputy,
                green);
        nodelabel('output3',
                adder^.outputx[3], adder^.outputy,
                green);
        nodelabel('output4',
                adder^.outputx[4], adder^.outputy,
                green);

        nodelabel('phi1',
                adder^.inclockx,  adder^.inclocky,
                red);
        nodelabel('phi2',
                adder^.outclockx, adder^.outclocky,
                red);

        enddef;

    wrapup;

end.  (** main program **)
```

This program retreives the definition of *adder* and creates a new symbol named *example* which contains an instance of it.

The **draw** command places uses the PLA's symbol name to place it at the origin. In the same fashion, connections to clocks, inputs, outputs, *Vdd* and *ground*, use the *adder* pointer to access the appropriate fields in the PLA attribute record. In this example, the inputs, outputs and clocks are labeled in this manner.

Appendix A

Details of NMOS macros "NEWPLA", "INPAD", "OUTPAD", and "PADTRISTATE".

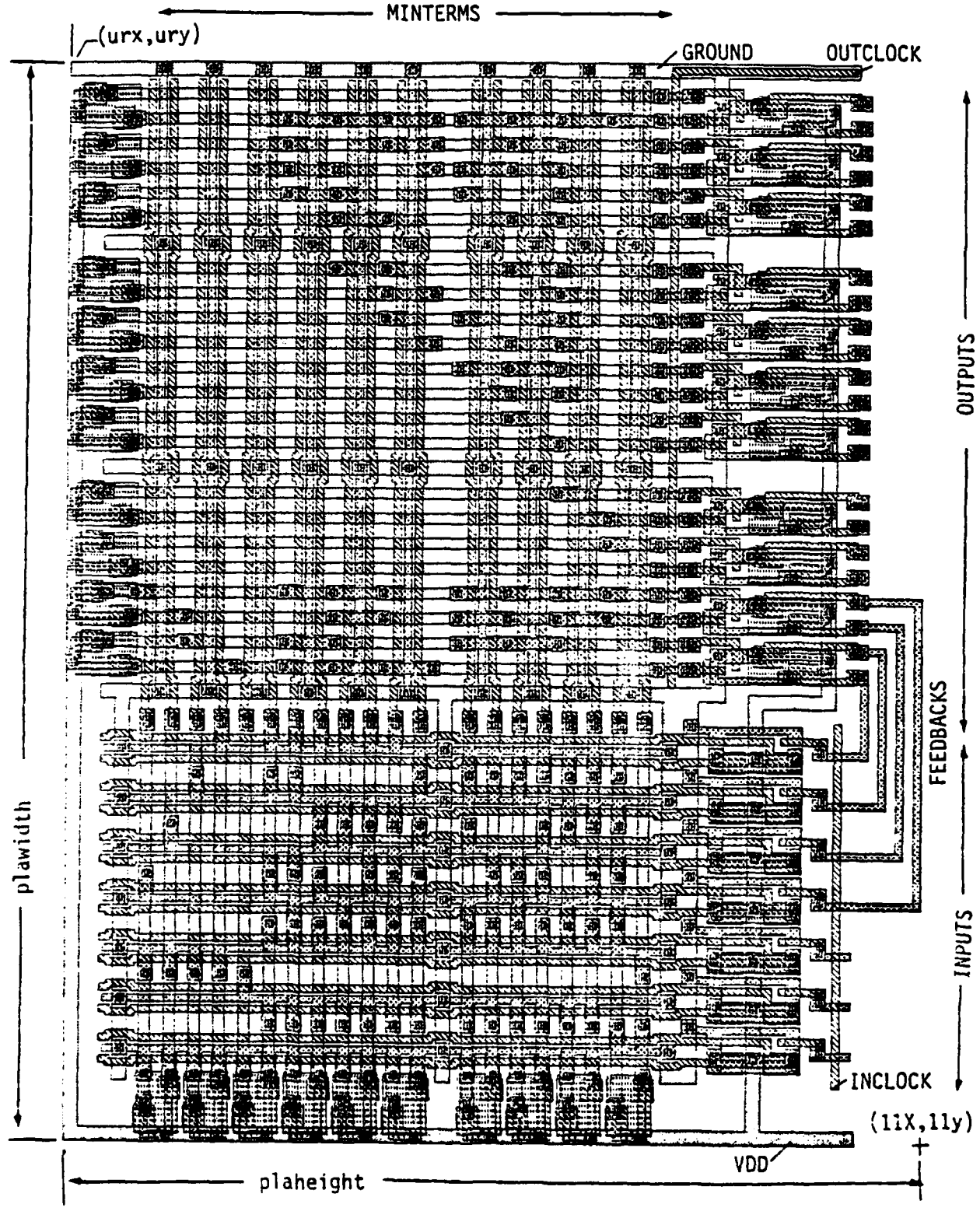


Figure 1 - PLA example, rotated 90^{o} ccw.

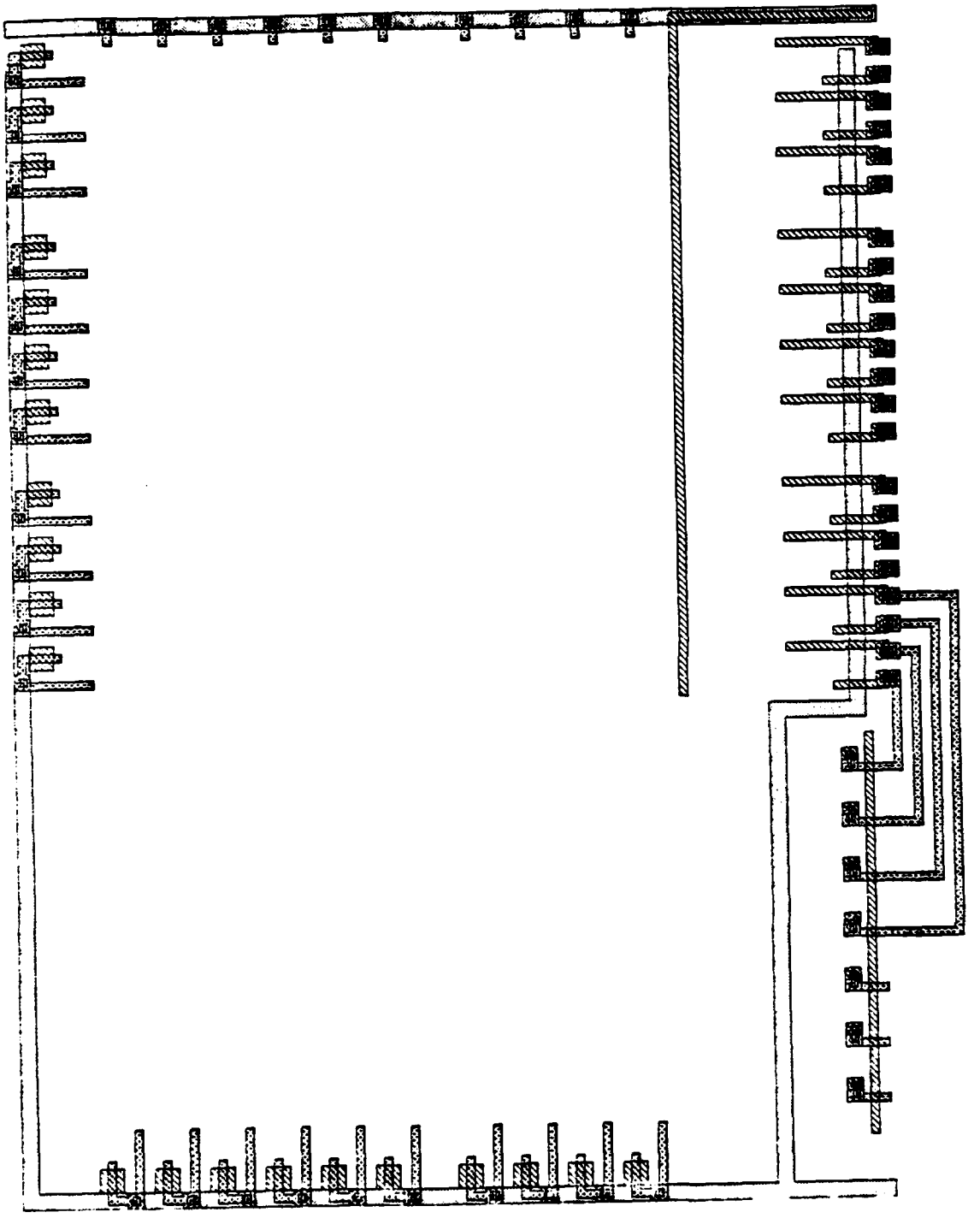


Figure 2 - Same PLA example but with Gutswitch set to "noguts". This option speeds up plotting.

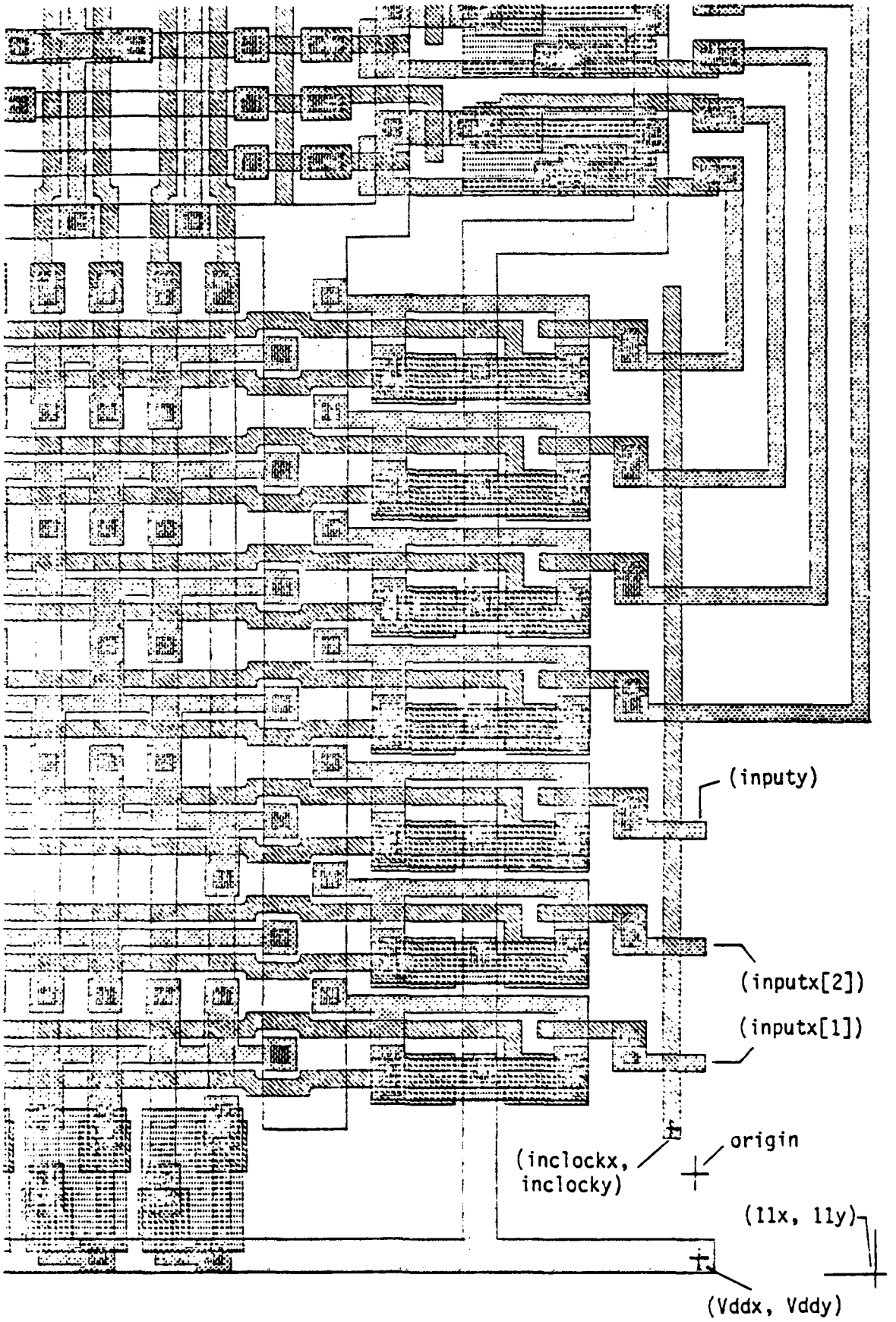


Figure 3 - Details of PLA origin and nearby critical coordinates. Input spacing is 14 lambda.

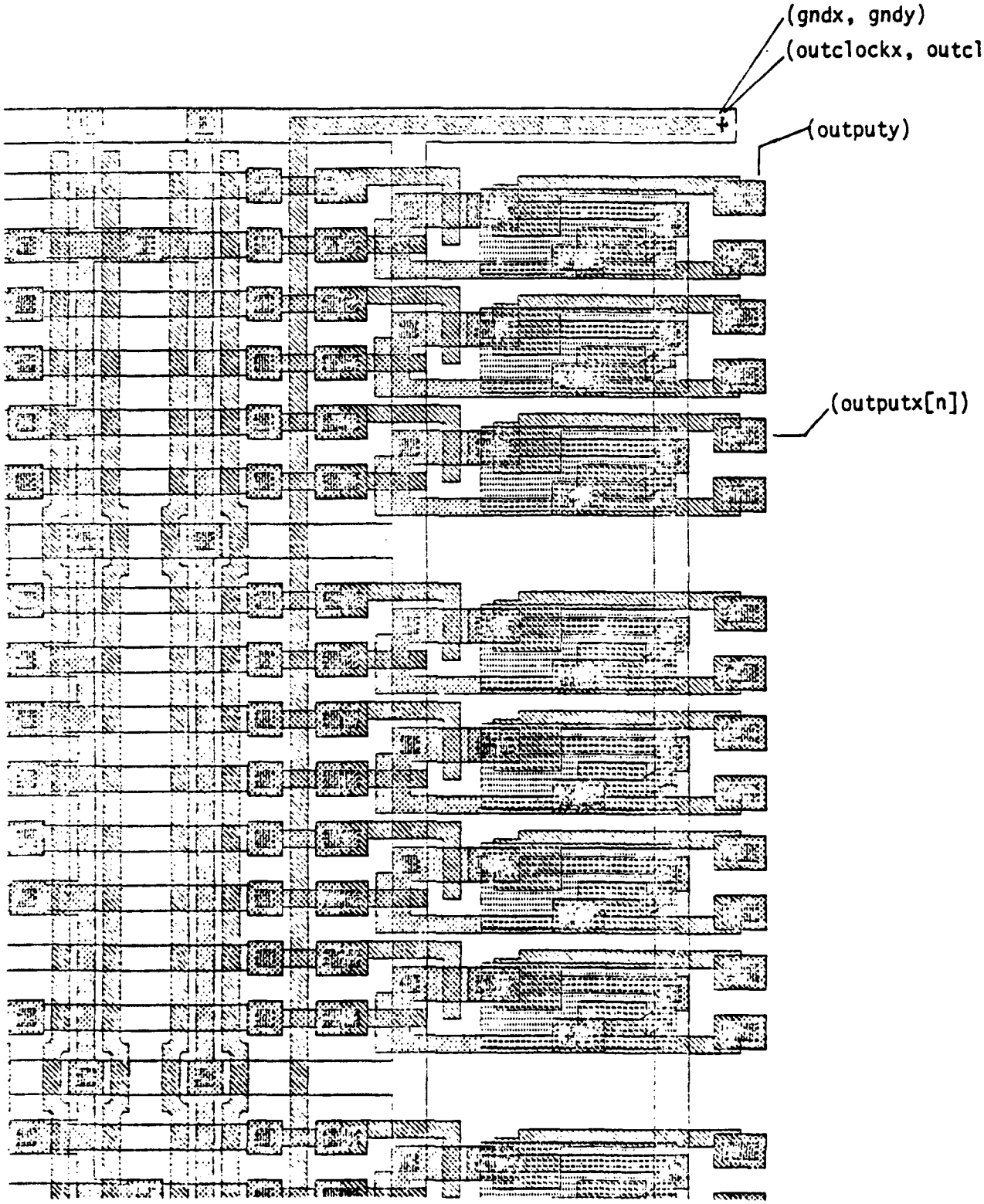


Figure 4 - Details of other critical PLA coordinates. Output spacing is 7 lambda, except for 14 lambda gaps.

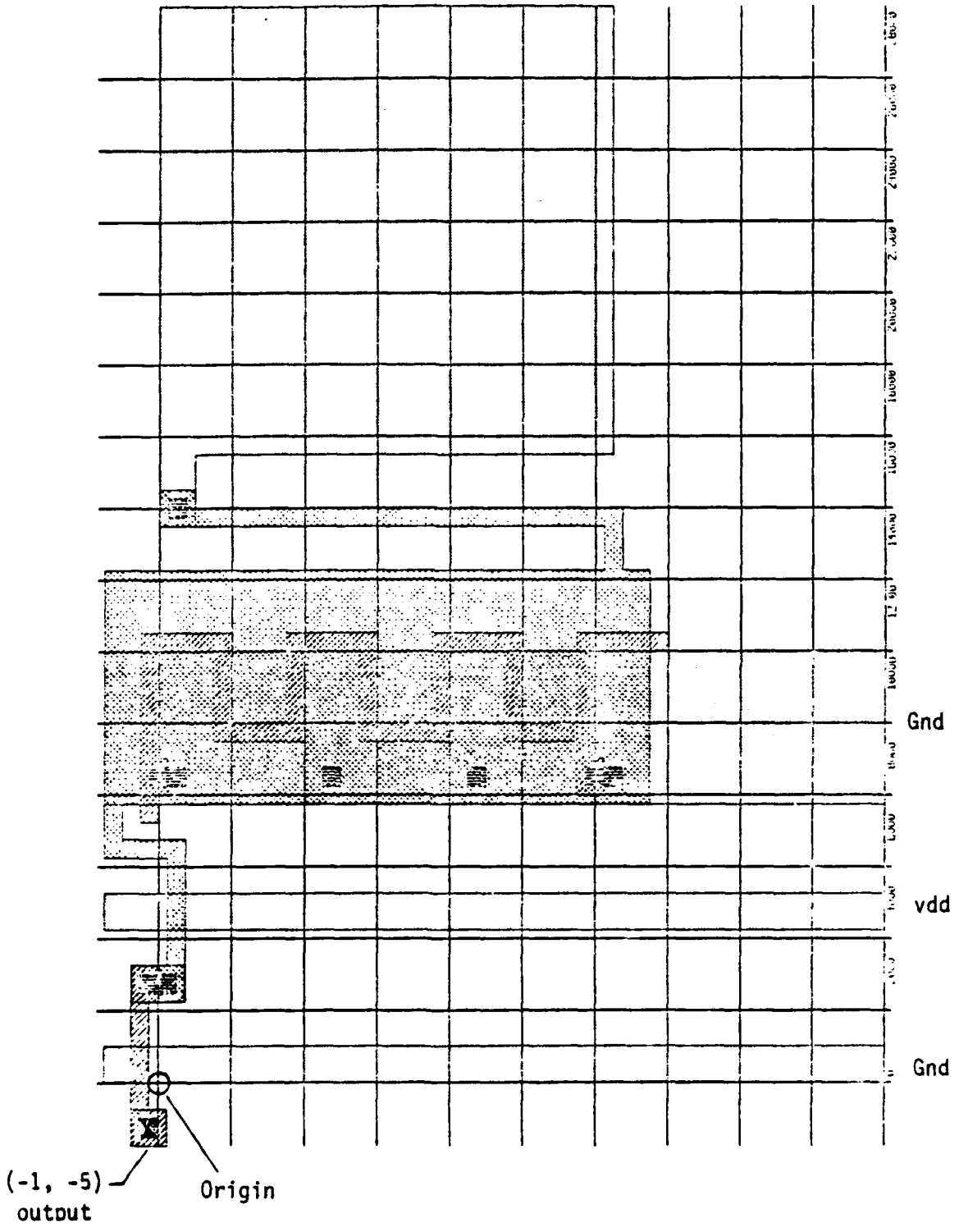


Figure 5 - Inpad (80, 0). Grid is 8 lambda.

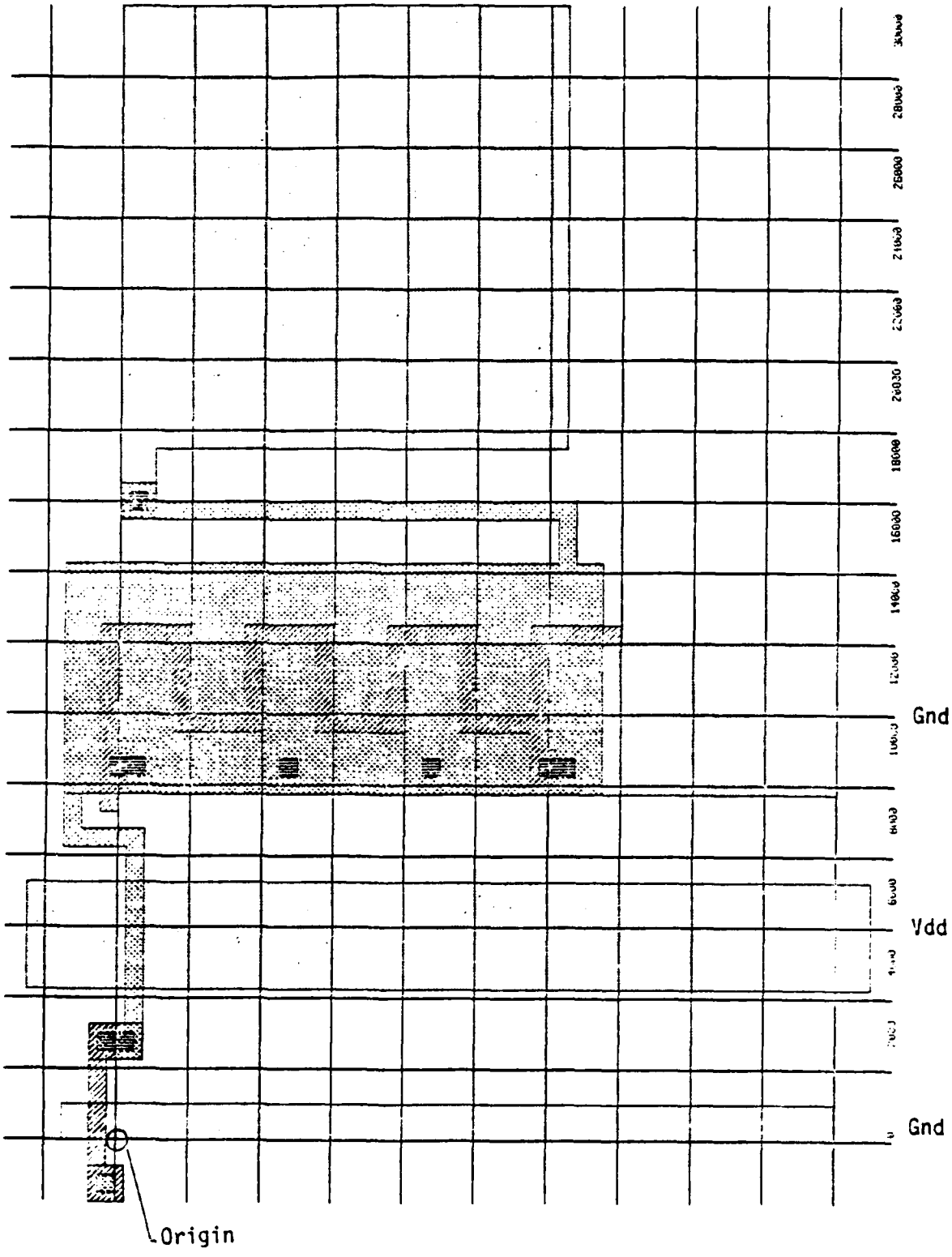


Figure 6 - Inpad (80, 8). Setting width = 8 fattens the Vdd bus by 8 lambda. Grid is 8 lambda.

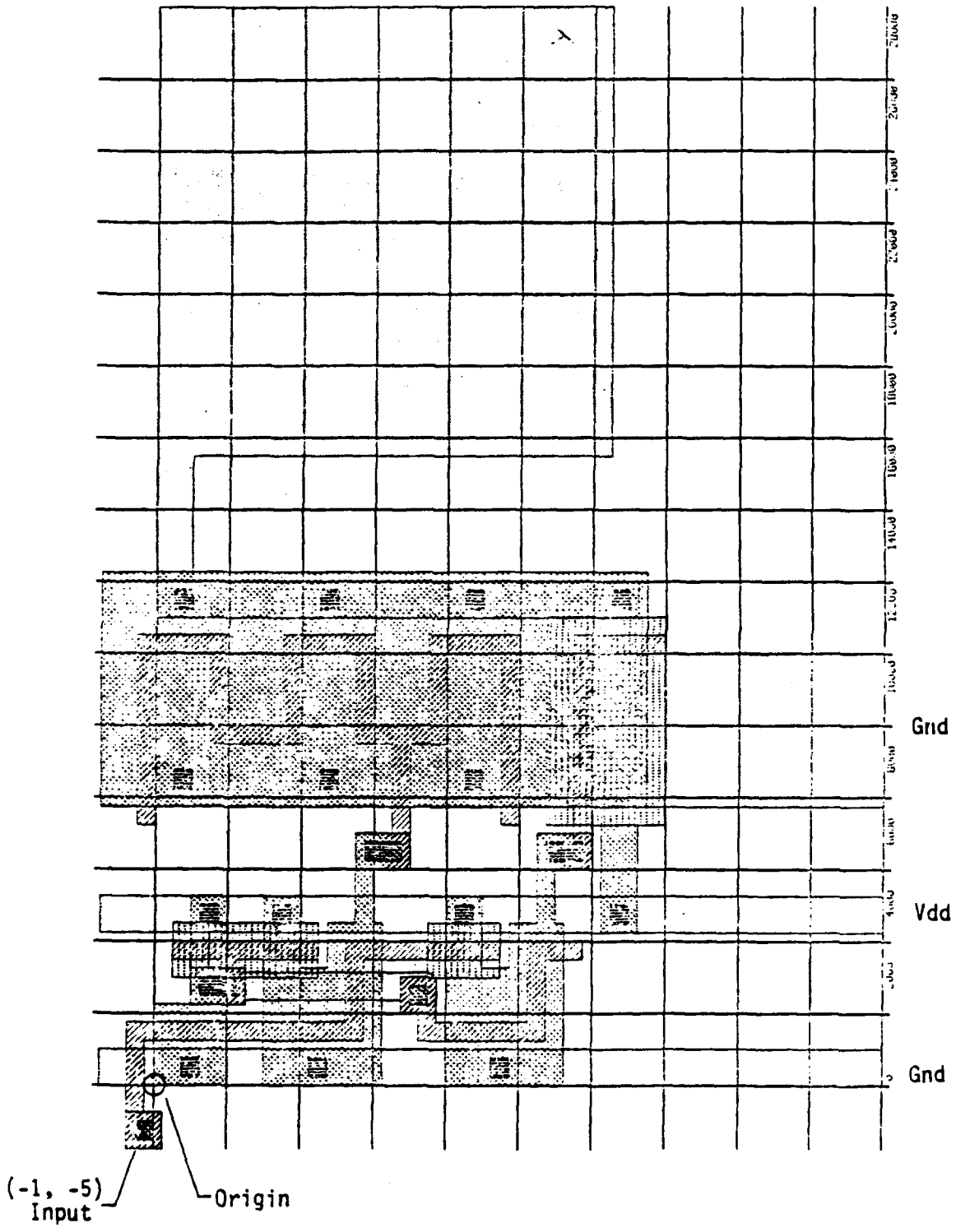


Figure 7 - Outpad (80, 0). Grid is 8 lambda.

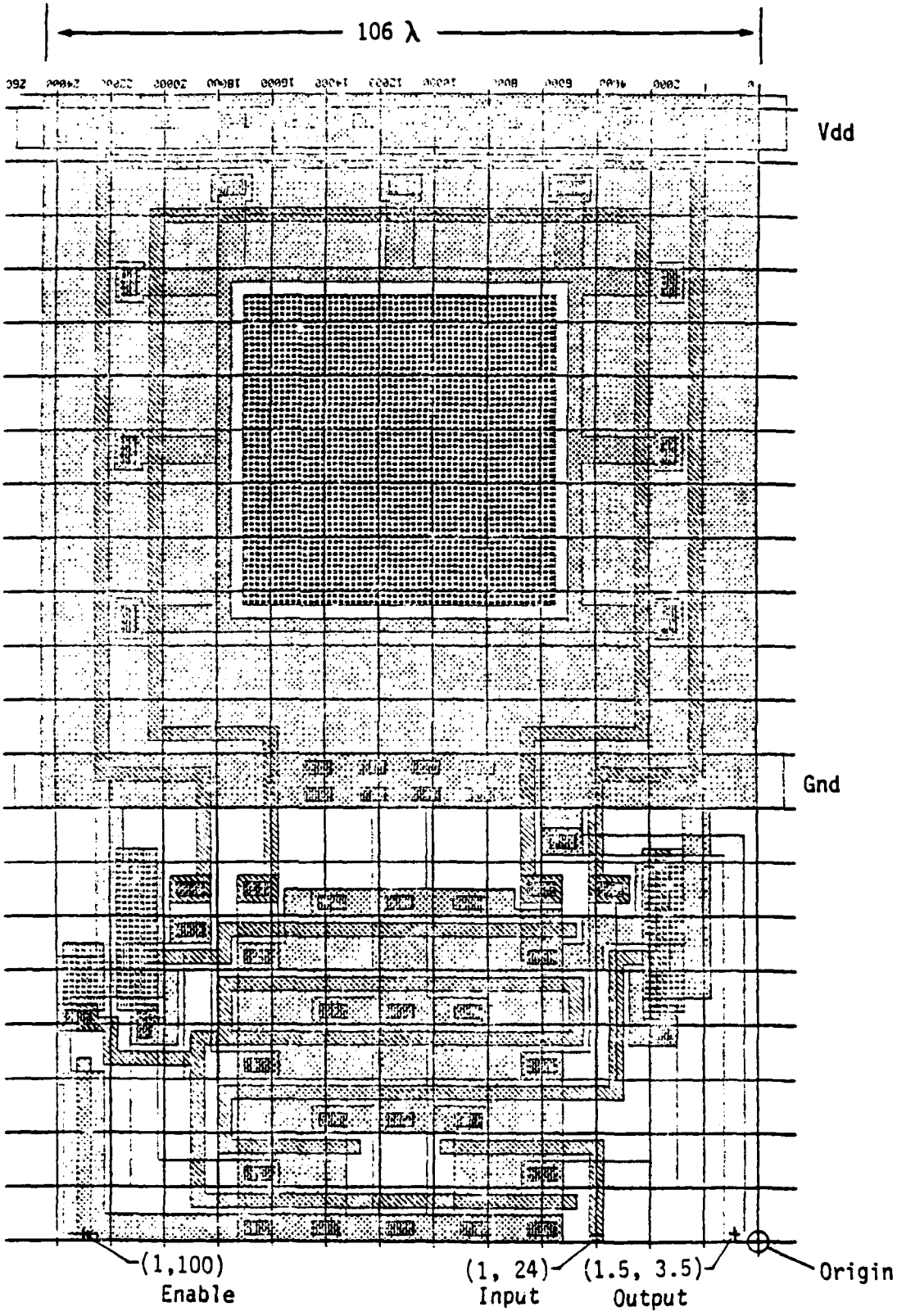


Figure 8 - Padtristate (4), rotated 90° CCW. 8 lambda grid. The "4" specifies now far the Gnd and Vdd tabs stick out.

RNL User's Guide

Christopher Terman

M.I.T. Laboratory for Computer Science
545 Technology Square, Room 418
Cambridge, MA 02135

1.0 INTRODUCTION

RNL (NetLisp) is a timing logic simulator for digital nMOS circuits with a lisp-like command interpreter. To use RNL, one needs a the mask file (e.g., CIF) or developed using NETLIST, a program that processes textual schematics.

RNL uses a resistive model of transistors to implement a logic-level timing simulator. See Chapter 2 of "Simulation Tools for LSI Design" by C. Terman for details of the algorithm.

A companion program, NL, is identical to RNL except that it implements a switch-level transistor model. NL is somewhat faster but does not correctly handle circuits that depend on the relative sizes of transistors for correct operation (i.e. there are circuits for which NL and RNL will get different answers). However, NL does work for "Mead and Conway" type circuits (so called ratioless logic) -- we have used it with good results on almost all the circuits to come out of MIT in the last couple of years.

One must first convert the .sim file to a network file suitable for use by RNL -- to do this we run PRESIM:

presim foo.sim foo [config]

which converts the file foo.sim into a binary file for RNL called foo. See PRESIM documents for information on the various configuration parameters.

To invoke RNL, just type

rnl

or perhaps

rnl cmdfile

if RNL is to initially read commands from the file cmdfile. Usually one prepares a command file that reads in the network from the .sim file and also loads a library file of RNL functions. As simulation proceeds, user defined functions developed for testing the circuit can be added to the command file. Thus, initially, the command file might contain the commands

```
(load "nl.l")
(read-network "foo")
```

You'll probably add some directory to "nl.l", the file foo was prepared by PRESIM.

When the end of the command file is reached, input is taken from the standard input. If you just wish to use RNL to poke around your network, after giving the two commands above, try the commands listed in the last section of this document.

2.0 INPUT

The top level of RNL is a simple loop:

(1) read command from current input;

(2) evaluate command, performing specified actions;

(3) print the result and loop back to (1).

There are two ways of specifying a command. The most general looks like

```
(function argument argument ... argument)
```

i.e., a list of names, numbers, etc. enclosed in parentheses. The parentheses delimit the command, so that white space (spaces, tabs, newlines) can be used to format the input any way one pleases. Of course, the arguments themselves may be of the form (function arg ... arg). The reader first reads the entire command -- up to the closing parenthesis. The arguments are then evaluated in left-to-right order and the results passed to function. The value returned by the function is printed and the reader invoked once again. For example, given the following input

```
(* 17 (+ 3 2)
(/ 10 2))
```

RNL would respond by typing 425 and then wait for more input. Note that nothing happened after the first newline since the first parenthesis had not yet been closed.

An alternative form for a command is

```
function argument argument ... argument <newline>
```

This is equivalent to

```
(function '(argument argument ... argument))
```

The "'" is explained below. The alternative form is useful for invoking simple simulator functions without getting all bolluxed up in parenthesis. Comments can be included by preceding them with ";" -- all characters following the ";" up through the next newline are ignored.

3.0 OBJECTS and VALUES

Here's a list of the objects that RNL knows about

numbers -- signed integers. (16 bits on PDP11s, 24 bits on VAXen, 28 bits on PDP10s).

-- floating point.

strings sequence of characters enclosed in quotes ("). Useful as constants for file names, print statements, etc. Special characters can be introduced into the strings by using the backslash escapes:

\n' newline

\r' return

\t' tab

\ooo' ascii code "ooo" where ooo are octal digits

symbols — variable names. Any sequence of characters that isn't a number, string, or some special character -- starting symbols with a letter, followed by more letters, numbers, and punctuation is usually a safe bet.

nodes — an electrical node.

transistors — a mosfet with gate, source, and drain.

lists — a sequence of objects enclosed in parentheses. Standard LISP syntax applies, including dot notation. The empty list "()" is also called "nil".

subrs — primitive, or built-in, functions (like +).

One can evaluate an object for a value; numbers, strings, subrs, nodes, and transistors are "self-evaluating", i.e., the object and its value are one and the same.

Evaluating a symbol yields the value last assigned to that symbol by the user (see the setq function). Symbols actually have two distinct values: the value used during evaluation and one used only when the symbol is used as a function name. This second value is ordinarily of no concern to the user and is used to associate symbol names with built-in functions.

Evaluating a list is like making a function call. The value of the first element of the list is the function, values of the remaining list elements are the arguments. For example, evaluating

```
(+ a 3)
```

looks up the value of the "+" symbol (in this case it will be the subr for addition), then calls the function with the values (recursively computed) of "a" and "3". The value of the list will be the value returned by the function.

Certain lists have special meaning to the system and are called special forms. Two special forms of particular interest are discussed here, the remainder are described in the following section. The quote special form,

```
(quote arg)  or  'arg
```

allows us to create symbol and list constants. Thus the value of (quote a) is the symbol "a", and the value of '(+ 2 3) is a list of three elements.

User defined functions are represented by the lambda special form:

```
(lambda (param param ...) exp exp ...)
```

The symbol "lambda" indicates that this list is actually a user function. It is followed by a list giving the names of the arguments and finally by a sequence of expressions which make up the body of the function. The value returned by the function when called will be the value of the last expression in the body. For example,

```
((lambda (x) (+ x 3)) 4)
```

evaluates to 7. We can give this function a name by making the lambda expression the value of some symbol:

```
(setq plus-3 '(lambda (x) (+ x 3)))
(plus-3 4)
```

also evaluates to 7. We usually write the first expression as

(defun plus-3 (x) (+ x 3))

which makes (lambda (x) (+ x 3)) the function definition of the symbol "plus-3". Note that setq changes the "expression value" of a symbol, while defun changes the "function value" -- this distinction is unimportant in most applications, but is useful if you wish to change the definition of a built-in function (beware of the implications before trying to change built-in function definitions though!).

4.0 BUILT-IN FUNCTIONS

The functions are listed by application area. The areas are:

arithmetic functions
predicates
list functions
i/o functions
miscellaneous functions
special forms
network/simulation functions
functions defined in nl.l

Unless otherwise stated all functions evaluate their arguments. The last section describes some library functions that provide a simple interface directly to the simulator. The library file can also serve as an example of user-defined functions for the curious.

4.1 Arithmetic functions

Unless otherwise stated, the arithmetic functions take both floating point and integer arguments, returning a floating point result if any argument was a floating point number.

(+ num1 num2 ...)

returns the sum of its integer arguments.

(- num1 num2 ...)

returns num1 minus the remaining arguments. (- num1) just evaluates to the negative of num1.

(* num1 num2 ...)

returns the product of its integer arguments.

(/ num1 num2)

returns the integer result of dividing num1 by num2.

(% num1 num2)

like division, but returns remainder (integers only)

(1+ num)

like (+ num 1) but for integers only.

(1- num)

like (- num 1) but for integers only.

(max num1 num2 ...)

returns the maximum of its integer arguments.

(min num1 num2 ...)

returns the minimum of its integer arguments.

(abs num)

returns the absolute value of its integer argument.

(fix num)

converts num to an integer.

(float num)

converts num to a floating point number.

4.2 Predicates

(< num1 num2)

less than

(< = num1 num2)

less than or equal to

(== num1 num2)

equal to

(!= num1 num2)

not equal to

(> = num1 num2)

greater than or equal to

(> num1 num2)

greater than

compares the two integer arguments; returns the symbol t if the predicate is satisfied, nil otherwise.

(not exp)
(null exp)

returns nil if exp is non-nil value; returns t otherwise.

(eq exp1 exp2)

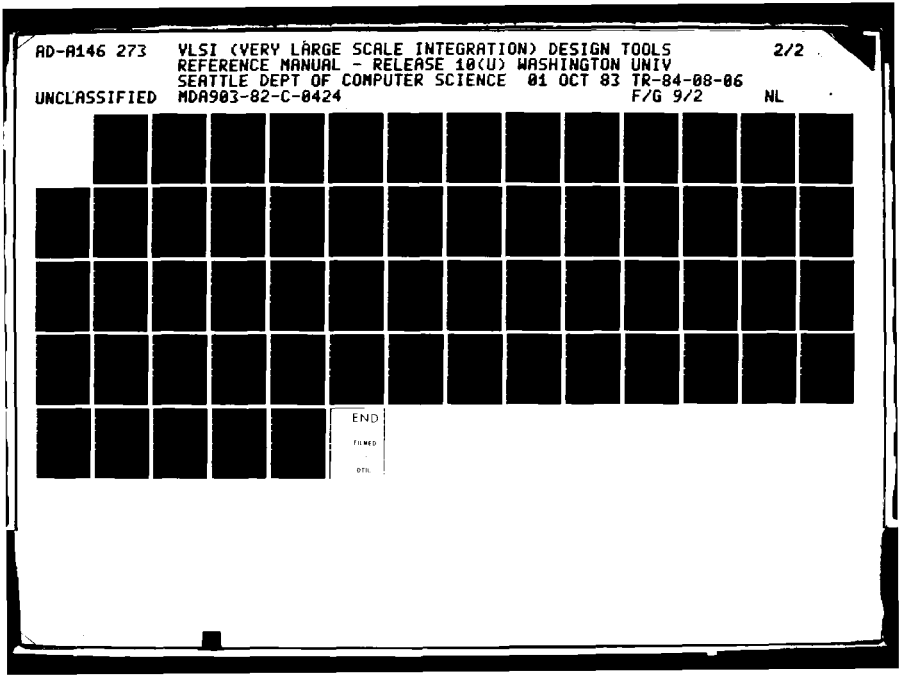
returns t if exp1 and exp2 are identical; nil otherwise. See equal.

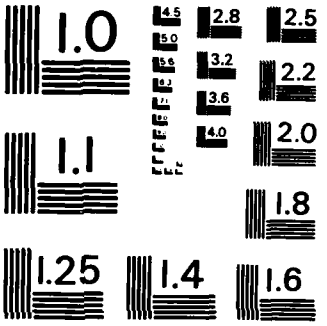
(equal exp1 exp2)

returns t if the exp1 and exp2 are conformable; nil otherwise. (eq x y) is equivalent to (equal x y) except when x and y are lists or strings. When x and y are lists equal returns t if corresponding list elements are equal (a recursive defintion). No one much cares about strings...

(memq exp1 exp2)

checks if the exp1 is "eq" to an element of the list exp2. If so, returns that portion of exp2 whose first element is exp1; otherwise nil is returned.


AD-A146 273
VLSI (VERY LARGE SCALE INTEGRATION) DESIGN TOOLS
REFERENCE MANUAL - RELEASE 10(U) WASHINGTON UNIV
SEATTLE DEPT OF COMPUTER SCIENCE 01 OCT 83 TR-84-08-06
UNCLASSIFIED
MDA903-82-C-0424
F/G 9/2
NL
2/2
END
FILMED
DTIC



(atom exp)

returns nil if exp is a list; t otherwise.

(numberp exp)

returns t if exp is a number; nil otherwise.

(stringp exp)

returns t if exp is a string; nil otherwise.

4.3 List functions

(car l)

returns the first element of the list l.

(cdr l)

returns a list containing all but the first element of l.

(rplaca l exp)

changes the car of l to be exp and returns (the modified) l.

(rplacd l exp)

changes the cdr of l to be exp and returns (the modified) l.

(cons exp1 exp2)

makes a list whose car is exp1 and whose cdr is exp2.

(list exp1 exp2 ...)

makes a list with elements exp1, exp2, ...

4.4 I/O functions

The i/o routines all take an optional fid argument which specifies what file to use -- if fid is ommitted standard input and output are used, as appropriate.

The base used for printing integers is controlled by the value of the symbol "base" -- default is decimal, base 10.

(read)
(read fid)

read from the specified input stream and return the next lisp object found. If the end of the input stream is reached, return the symbol end-of-file.

(printf "format..." exp exp ...)
(printf fid "format..." exp exp ...)

formatted output. The string "format..." tells how the output should look: characters are printed directly as given in the string until an escape (the characters "%S") is found -- the escape sequence is replaced by the print representation of the next unused exp argument. For example,

```
(setq a 3 b 4)
(printf "The sum of %S and %S is %S\n a b (+ a b))
```

would output

The sum of 3 and 4 is 7< newline>

The possible escapes are of the form %* and are the following

"%c" output the fixnum value of the next exp as a character.

"%%"output the character '%'

"%S" output the value of the next exp (any lisp object)

(print exp)
(print exp fid)

print exp followed by a newline character.

(prin1 exp)
(prin1 exp fid)

just like print without the terminating newline.

(princ exp)
(princ exp fid)

just like prin1 except strings are printed without the surrounding quotes. Useful for printing messages using string constants for the text.

(terpri)
(terpri fid)

output a newline.

(openi exp)

opens the file whose name is the value of exp for input. The value returned by openi can be used as an argument to read and close.

(openo exp)

opens the file whose name is the value of exp for output. The value returned by openo can be used as an argument to the various print routines and close.

(flush fid)

force out any buffered output for the specified stream.

(close fid)

close the file specified by fid (fid was the value returned by either openi or openo).

(load exp)

starts reading rnl commands from the file whose name is the value of exp. Returns when end-of-file is reached.

4.5 Miscellaneous functions

(set expv exp)

evaluates the first argument to get a symbol whose value becomes the value of the second argument. Useful, but see setq below. returns the value of exp.

(fset expv exp)

like set, but changes the function value.

(symeval sym)
(fsymeval sym)

returns the value of the symbol "sym" in the current environment. This function is only useful for getting the value of a symbol whose name you don't know until run time. fsymeval is similar but returns the function value named by the symbol.

(eval exp)

returns the value of exp, i.e., exp is evaluated twice. This function is useful for finding the value of expressions constructed at run time by the program. For example, (eval '(+ 3 4)) = 7.

(make-symbol exp1 exp2 ...)

returns a symbol whose pname is the concatenation of the print representations of exp1, exp2, ... Useful for programmatically constructing names of nodes.

(exit)

exit RNL, returning to system command level.

4.6 Special forms

Although special forms look like function calls, their behavior especially as regards the evaluation of their "arguments" can be quite different.

'exp
(quote exp)

turns exp in a constant. See explanation in previous section.

(and exp1 exp2 ...)

evaluates the exp's from left to right stopping when the first null value (i.e. nil) is encountered. Returns value of last expression evaluated.

(or exp1 exp2 ...)

evaluates the exp's from left to right stopping when the first non-null value (i.e. something besides nil) is encountered. Returns value of last expression evaluated.

(setq var exp)

makes the value of exp the new value of the variable var. Note that exp is evaluated while var is not.

(defun name (var1 var2 ...) exp exp ...)

makes the function definition of the symbol "name"

(lambda (var1 var2 ...) exp exp ...)

See discussion about functions in previous section.

```
(cond (pred1 exp exp ...)
      (pred2 exp exp ...)
 ...)
```

extended conditional construct. Following the cond is a sequence of clauses, in each clause the first expression is taken to be a predicate and the remainder of the expression the body of the clause. Cond evaluates the predicates in turn, stopping at the clause with the first non-nil value for the

predicate. The body of that clause is then evaluated and the value of the last expression in the body is returned as the value of the cond. If no predicates evaluate to non-nil, nil is returned as the value of the cond.

(prog (var1 var2 ...) exp exp ...)

The old values of var1, var2, ... are saved away and each variable is given a new value of nil. Then each expression in the body is is evaluated, and the value of the last returned as the value of the prog. When the prog is exited, the old values for the vars are restored. Use prog's to allocate local variables.

(do (iter1 iter2 ...)
(exit eexp eexp ...)
bexp bexp ...)

extended iteration construct. The do has three basic parts:

(i) iteration variable declaration

(ii) exit test and expressions

(iii) the body.

The index variable declarations allow one to declare temporary variables, specifying their initial values and how they are incremented at each iteration. The iter's have the form

(var) or (var init) or (var init incr)

where init and incr are optional. var is the name of this index variable. init is evaluated to give its initial value; if not present nil is used. incr is evaluated at the end of each iteration to give the next value of the index; if omitted the variable is not changed automatically. The following iterator declares an index which starts at zero and is incremented by two:

(i 0 (+ i 2))

Before each iteration, the exit test is evaluated. If its value is non-nil, the eexp's are evaluated in turn and the value of the last returned as the value of the do. If its value is nil, the bexps are evaluated, and the process repeated.

4.7 Network functions

An electrical network consists of a list of nodes and transistors. In RNL, you can deal with only one network at a time. The functions described in this section allow user-defined functions to deal with the network; the following sections describes certain library functions which the user may wish to use when typing in commands one-by-one.

(read-network "filename")

reads a network file from "filename" and adds it to the current network. There is no way to re-initialize the network, except by restarting RNL.

(find-node nodename)

given a node name returns the corresponding node object. All the functions that deal with nodes take either a name or the object itself, so this function isn't really necessary.

(match-node pattern)

given a pattern returns a list of nodes whose name match the pattern; "*" wild cards are allowed.

(node-value node)

returns the current value of a node: either 0, 1, or X.

(set-node node exp)

adds/removes node as an input. If exp evaluates to

0	node is added to low input list
1	node is added to high input list
U,u	node is added to undefined input list
X,x	...see text...

and this node will be stuck at this input value until changed by another call to set-node. If exp is X (remember exp is evaluated so you'll probably want to type 'X), node is removed from the input lists -- at the next simulation step it will acquire whatever value it would naturally have.

(set-threshold node vlow vhigh)

set the logic thresholds for the specified node; vlow and vhigh should be numbers in the range [0,1]. These thresholds are used when converting from a Thevenin equivalent voltage to a logic state -- sometimes it's useful to be able to override the defaults for special nodes which otherwise will turn out X.

(set-delay node tplh tphl)

set the transition times for the specified node; tplh (low-to-high transition time) and tphl (high-to-low transition time) are integers specifying time in 10ths of nanoseconds. If either tplh or tphl are negative, the node's times become unspecified and the transition times will be determined by the usual RC calculation. This command allows one to override the timing calculation -- useful when it's getting the wrong answer for one reason or another (usually this is worth doing only on critical nodes, such as clocks, where an timing error can be significant).

current-time

The value of the symbol current-time gives the current simulation time in tenths of nanoseconds. Value is a flonum.

(sim-step stop-time node-list)

After using set-node to establish the value of the network's inputs, sim-step can be used to propagate the input values to the remainder of the network. stop-time specifies at what simulated time this simulation step should end; if sim-step returns nil, this time has been reached. node-list is a list of nodes -- if any of these nodes change during simulation, the simulation is step is aborted and sim-step returns the name of the node whose value changed (to restart the simulation where you left off, just make another call to sim-step with the same arguments as the first time -- see step procedure in the final section below).

(switch-step stop-time node-list)

just like sim-step, except transistors are modelled as switches and transitions have unit delay. This algorithm is somewhat faster than the usual RNL calculation for many circuits, but can give X answers for circuits for which transistor size is important for correct operation (e.g., bit line in a dynamic memory). To ensure correct operation, one should not use sim-step until the event list is empty (and vice versa) -- i.e., all events scheduled by a particular algorithm must be handled by the same algorithm; the value of event-list-empty can be tested to see if the all events have been handled.

This routine may be useful when debugging the basic functionality of a circuit, or when simulating a circuit which has not been correctly sized (one that gives ratio errors using sim-step). Since the

switch-level algorithms are much faster when dealing with large groups of interconnected nodes, switch-step may be particularly useful when initializing a network.

event-list-empty

This symbol has the value t if the event list is empty, nil otherwise -- updated after every sim-step.

(set-params name value)

Give a value to one of the simulation parameters:

report	flag (default = t). If non-nil, nodes given the value of X because of improper pullup/pulldown ratio or because of charge decay will cause a warning message to be printed.
unitdelay	flag (default = nil). If non-nil, all node transitions happen with unit delay.
decay	fixnum (default = 0). If non-zero, tells the number of time units (10ths of nanoseconds) it takes for charge on a node to decay to X. A value of 0 implies no decay at all.
maxres	number (default = 1E10). Capacitors on the far side of transistors bigger than this value don't contribute to summary capacitance used in calculating transition times.

? node1 node2 ...
(? nodelist)

lists transistors whose source or drain are connected to the specified nodes. This command is useful for wandering through the network trying to track down the source of a particular value.

! node1 node2 ...
(! nodelist)

lists transistors whose gates are connected to the specified nodes.

(sim-init)

finds all X nodes and sets them to 0; running a simulation step (via s or step, etc.) will propagate the consequences through the network. The integer returned tells number of nodes that were changed; when this is zero initialization is complete. Usually a couple of calls to sim-init/step are needed; if there are still X nodes after, say, three or four calls, sim-init probably can't initialize the network.

(switch-init)

like sim-init, except prepares network for initialization by switch-step instead of sim-step.

(trace-node nodename t)
(trace-node nodename nil)

prints information about node during simulation step. Useful for trying to track down exactly what is happening to a subciruit at a very low level. The first form turns on tracing for the specified node, the second turns it off. The printout looks like:

```
[1]    ; event 1: count=H @ 123.3 ns
[2]    ;  count => clist: 13 10 <input seen>
[3]    ;  13: rgnd=[0+,0+], rvdd=[1,1] => value=L
[4]    ;  10: rgnd=[0+,0+], rvdd=[1,1] => value=L
[5]    ;   enqueuing 10 [event 1: count] new value=L
[6]    ;  count => clist: 11 <input seen>
[7]    ;  11: rgnd=[0+,0+], rvdd=[1,1] => value=L
[8]    ;   enqueuing 11 [event 1: count] new value=L
[9]    ; event 2: ...
```

[1]: start of a new event, this event is calculating effects of count becoming a logic high (H).

[2]: the value of count affects nodes 13 10, which are electrically connected to each other and to an input node (probably vdd or gnd). These nodes form a connection list (clist).

[3,4]: the new value of each element on the clist is calculated. rgnd and rvdd are the effective resistances to gnd and vdd. Each resistance is actually an interval -- for example, an interval of [0+,inf] is used to represent the resistance of an enhancement mode transistor with X on its gate. Possible resistances are

0+ = short circuit through "on" transistor,

1 = connection by depletion device (e.g., a pullup),

inf = no connection at all.

[5]: node 10 changed value and is added to the end of the event list so the effects (if any) of the change will be calculated.

[6-8]: node 11 is also affected by count...

[9]: next event is fetched from beginning of event list, and so on.

(trace-all-nodes t)
(trace-all-nodes nil)

like trace-node for all nodes in the circuit. Sometimes this is the only way to track down oscillating subcircuits.

(write-state filename)

create the file "filename" and write the current state of the network into the file. see read-state.

(write-ascii-state filename)

create the file "filename" and write the current state of the network into the file -- like write-state except the output is meant to be read by the designer.

(read-state filename)

opens "filename" for input and restores the previously saved state of the network. Note only node values are saved/restored; the user is responsible for setting inputs to appropriate state. A rudimentary check is made to see if the state file was written using the same network; if the check fails you will get the message "state file has wrong number of nodes".

5.0 Functions defined in NL.L

h node1 node2 ...
(h nodelist)

makes the specified nodes logic high (1) inputs.

l node1 node2 ...
(l nodelist)

makes the specified nodes logic low (0) inputs.

u node1 node2 ...
(u nodelist)

makes the specified nodes undefined (X) inputs.

x node1 node2 ...
(x nodelist)

removes the specified nodes from the input lists -- the simulator will determine the nodes natural value on the next sim-step.

t node1 node2 ...
(t nodelist)

traces node1, node2, ... (see trace-node above).

ut node1 node2 ...
(ut nodelist)

stops tracing node1, node2, ... (see trace-node above).

c nsteps
(c nsteps)

performs 4 steps, running a two-phase non-overlapping phi1/phi2 clock cycle. nsteps specifies the number of complete clock cycles to do.

s
(step incr nodes)

run a simulation step until time current-time+incr. "incr" and "nodes" are two global variables specifying how long simulation step should be (in tenths of nanoseconds) and which node transistions are to be reported. What sort of simulation algorithm is used is determined by the value of the symbol switch-level:

switch-level = tuse switch-step (switch-level, unit delay)

switch-level = niluse sim-step (resistor model, RC timing)

w node1 node2 ...

adds the specified nodes to the list of watched nodes. If node1 is of the form

(bit name node1 node2 ...)

the node will be displayed as a vector of bits (e.g. name=10X1). In addition to bit, there is also dec (decimal).

uw node1 node2 ...

un-watch the specified nodes.

PRESIM User's Guide

Christopher Terman

M.I.T. Laboratory for Computer Science
545 Technology Square, Room 418
Cambridge, MA 02135

One must first convert the .sim file to a network file suitable for use by RNL or NL -- to do this we run PRESIM:

presim foo.sim foo [*config*] *options...*

which converts the file foo.sim into a binary file for RNL/NL called foo.

The -c option:

-cfile,minvalue

writes a list of node names and capacitances to the specified file. Only capacitances larger than minvalue will be included.

The -t option:

-tfile,minvalue

writes a list of transistors and RC values to the specified file -- there are two entries for each transistor. The R's come from the size of the transistor, C's from the source/drain capacitance. Only RC values larger than minvalue will be included.

The -p option:

-presist,voltage

provides a worse-case estimate of the circuit power consumption by assuming that all the pullups (DEP or LOWP devices with drain=VDD) are all on simultaneously. "Voltage" specifies the supply voltage, for example "-p5" specifies a VDD of 5 volts. The result is printed after PRESIM completes its other processing. When figuring the resistance of a pullup device the "power" characteristic resistance as set in the config file is used.

The optional third file (config) specifies various electrical parameters; the format of this file is lines of the form

parameter value comments...

Lines beginning with ";" are treated as all comment. The parameter names and their default values

are:

```
; configuration file for "standard" MPC process

capm2a .00000  ; 2nd metal capacitance -- area, pf/sq-micron
capm2p .00000  ; 2nd metal capacitance -- perimeter, pf/micron
capma  .00003  ; 1st metal capacitance -- area, pf/sq-micron
capmp  .00000  ; 1st metal capacitance -- perimeter, pf/micron
cappa  .00004  ; poly capacitance -- area, pf/sq-micron
cappp  .00000  ; poly capacitance -- perimeter, pf/micron
capda  .00010  ; n-diffusion capacitance -- area, pf/sq-micron
capdp  .00060  ; n-diffusion capacitance -- perimeter, pf/micron
cappda .00010  ; p-diffusion capacitance -- area, pf/sq-micron
cappdp .00060  ; p-diffusion capacitance -- perimeter, pf/micron
capga  .00040  ; gate capacitance -- area, pf/sq-micron

lambda 2.5     ; microns/lambda (conversion from .sim file units
               ; to units used in cap parameters)

lowthresh  0.3 ; logic low threshold as a normalized voltage
highthresh 0.8 ; logic high threshold as a normalized voltage

cntpullup 0    ; <>0 means that the capacitor formed by gate of
               ; pullup should be included in capacitance of output
               ; node

diffperim 0    ; <>0 means do not include diffusion perimeters
               ; that border on transistor gates when figuring
               ; sidewall capacitance (*)

subparea 0     ; <>0 means that poly over transistor region will not
               ; be counted as part of the poly-bulk capacitor (*)
```

```
diffext 0       ; diffusion extension for each transistor, i.e., each
                ; transistor is assumed to have a rectangular source
                ; and drain diffusion extending diffext units wide and
                ; transistor-width units high. The effect of the
                ; diffusion extension is to add some capacitance to
                ; the source and drain node of each transistor --
                ; useful when processing the output of NET to improve
                ; the capacitive loading approximations without adding
                ; explicit load capacitors. diffext is specified in
                ; lambda (it will be converted using the lambda factor
                ; above).

resistance channel context width length resist
; this command specifies the equivalent resistance for a transistor
; of type channel with the specified width and length. Transistors
; matching this entry will have the specified resistance; linear
; interpolation is done if the width and/or length is not matched
; exactly.
;  channel is one of "enh", "dep", "intrinsic", "low-power",
;                    "pullup", or "p-chan"
; context is one of "static", "dynamic-high", "dynamic-low", or "power"
; width is given in lambda
; length is given in lambda
; resist is given in ohms
```

(*) These paramters should be 1 only when processing the output of the node extractor. They cause various corrections to be made to the interconnect component of a node's capacitance -- usually only extracted .sim files have information regarding interconnect capacitance.

PRESIM uses these parameters in calculating the capacitance for each electrical node and the resistance for each transistor channel.

NETLIST User's Guide

Christopher Terman

M.I.T. Laboratory for Computer Science
545 Technology Square, Room 418
Cambridge, MA 02135

1.0 Command Line

To run NETLIST type

netlist *infile* [*outfile*] [-o] [-s#] [-d#,#] [-e#,#]

infile is the name of the NETLIST input file, if outfile is specified, that file is used for output. The options are:

-o old format input. size specifications are taken to be length/width rather than width/length.

-s# use specified number as initializer for internal node names; useful when you want to merge the results of separate NETLIST runs.

-d#,# set the default width (first number) and length (second number) for depletion devices. The defaults are 8 and 2.

-e#,# like -d except for enhancement devices. The defaults are 2 and 2.

-i#,# like -d except for intrinsic devices. The defaults are 2 and 2.

-l#,# like -d except for low-power devices. The defaults are 2 and 2.

-p#,# like -d except for p-channel devices. The defaults are 2 and 2.

2.0 NETLIST Commands and Macros

A NETLIST file can insert other NETLIST files by using the include com- mand:

include filename.net

The single argument must be a string (i.e., enclosed in quotes).

Available through NETLIST are all the regular built-in functions of RNL (i.e., a subset of standard LISP primitivies) -- see RNL documentation for a description of these subroutines. In addition, NETLIST offers some spe- cial functions useful for building a description of a transistor network. These functions are described below.

NETLIST is a macro-based language for describing networks of sized transistors. Names in NETLIST refer to nodes, which presumably get inter- connected by the user through transistors. A node name has two forms

(n width length)

n is the name of the network node, length and width specify a transis- tor size. This is used in

NETLIST constructs where mention of a name causes creation of a transistor.

n

n is the name of the network node; when transistor sizes are required they are taken from the appropriate defaults.

When using a name to refer to a node, it must first be "declared" (this allows typo's to be caught early on). Nodes are declared by using the node statement or the local statement (see below). The node statement looks like:

(node n1 n2 n3 ...)

where n1, n2, etc are the names to be declared. Note: when using structured names (see the repeat statement) only the first component has to be declared.

The interconnect capacitance associated with a node can be speci- fied as follows:

(capacitance n 1.234)
(setq pf-sq-micron-of-diffusion 10e-4)
(capacitance n (* 13 pf-sq-micron-of-diffusion))

The first argument is the name of the node, the second the capacitance in pf (must be a number).

An electrical node can be given several names by using the con- nect statement:

(connect n1 n2 n3 ...)

The names n1, n2, etc. will all refer to the same electrical node. This statement is useful for connecting i/o signals to the edge of an array generated by a repeat statement.

The voltage threshold for logic high and low states can be set by the NETLIST command threshold:

(threshold n 0.2 0.8)

would set the logic low threshold for node n to 0.2 (normalized vol- tage) and the high threshold to 0.8. If no threshold is specified, the node will be given the default thresholds as given in the confi- guration file for PRESIM (see PRESIM.DOC for details).

The "delay" of a node (the transition times for changes in the node's value) can be specified by user with the delay command:

(delay n plh phl)

where plh and phl are integers specifying the low-to-high transition delay and the high-to-low delay respectively. Delays are specified in RNL time units (1/10th nanosecond). If you do not specify a delay for a node, RNL will calculate one based on the impedence of the driving transistors and the capacitance of the node; user-specified delays override the usual RNL calculation.

Node interconnections are accomplished by one of the following NETLIST statements:

(trans g s d [w [l]])
(etrans g s d [w [l]])

enhancement mode transistor with gate g, source s, and drain d. l and w specify length and width of transistor (can be ommited).

(dtrans g s d [w [l]])

like etrans, except depletion mode transistor

(itrans g s d [w [l]])

like etrans, except intrinsic transistor

(ltrans g s d [w [l]])

like etrans, except low-power transistor

(ptrans g s d [w [l]])

like etrans, except p-channel transistor

(pullup a)

depletion-mode pullup (to vdd) of a.

(pulldown a n-1 ... n-k)

chain of k transistors from a to gnd, gates of transistor are n-1,

(invert a b)

two-transistor inverter with output a and input b.

(nor a n-1 ... n-k)

pulls up a, and creates k transistors from a to gnd controlled by n-1 through n-k.

(nand a n-1 ... n-k)

equivalent to

```
(pullup a)
(pulldown a n-1 ... n-k)
```

(and-or-invert a (n-1 ... n-k) ... (m-1 ... m-l))

equivalent to

```
(pullup a)
(pulldown a n-1 ... n-k)
...
(pulldown a m-1 ... m-l)
```

Iteration construct is repeat statement:

```
(repeat index low high
    [(local l-1 ... l-j)]
    ...)
```

where index will be given successive values starting with low and fin- ishing with high. You can use the index in structured names, e.g.:

foo.index foo.(1+ index) foo.(1- index).bar ...

local variables are described under macros.

For ease of circuit entry, the user can build and call parameter- ized macros. macro definitions have the form

```
(macro n (p-1 ... p-k)
     [(local l-1 ... l-j)]
     ...)
```

where n is the name of the new NETLIST function being created, p-1 ... p-k are the formal parameters, l-1 ... l-j are the optional local node names used in the body.

The macro is invoked as follows:

(n a-1 ... a-k)

which causes the body to be interpreted after

1) all occurrences of p-1 in the body have been replaced by a-1, etc.

2) all occurrences of l-1 in the body have been replaced by a new, unique node name. Unique names will be a number (like for anonymous nodes in pulldowns).

3.0 Examples

In the following examples

```
e g s d l w
```

specifies an enhancement-mode transistor with gate g, source s, and drain d with length l and width w.

```
d g s d l w
```

is similiar, except transistor is depletion mode.

Quickie examples:

```
(invert a b)
d a a vdd 8 2
e b a gnd 2 2
```

```
(invert a (b 17 5))
d a a vdd 8 2
e b a gnd 5 17
```

```
(invert (a 2 2) (b 2 4))
d a a vdd 2 2
e b a gnd 2 4


(nor (a 16 2) (b 2 4) c d)
d a a vdd 2 16
e b a gnd 4 2
e c a gnd 2 2
e d a gnd 2 2


(and-or-invert a (b c d) (e f) (g))
d a a vdd 8 2
e b a 1001 2 2
e c 1001 1002 2 2
e d 1002 gnd 2 2
e e a 1003 2 2
e f 1003 gnd 2 2
e g a gnd 2 2
```

Two dimensional array of foo's:

```
(repeat i 1 8 (repeat j 1 8 foo.i.j))
```

generates

```
foo.1.1  foo.1.2  foo.1.3  ... foo.1.8
foo.2.1  ...                 foo.8.8
```

Simple two-inverter dynamic memory cell:

```
(macro bitcell (output output-enb input input-enb refresh)
    (local a b c)
    (trans input-enb input a 2 4)
    (invert b a)
    (invert (c 2 2) (b 2 8))
    (trans refresh a c)
    (trans output-enb c output 2 4)
)

(bitcell bit0 renb bit0 wenb phi2)
```

generates

```
e wenb bit0 1001 4 2
d 1002 1002 vdd 8 2
e 1001 1002 gnd 2 2
d 1003 1003 vdd 2 2
e 1002 1003 gnd 8 2
e phi2 1001 1003 2 2
e renb 1003 bit0 4 2
```

Assume you had an alu bit-slice macro of the following form

```
(alu carry-in operand1 operand2 result carry-out)
```

then the following macro would produce an n-bit alu:

```
(macro ALU (n databus1 databus2 resultbus cin cout)
     (connect cin cout.0)
     (repeat i 1 n
(alu cout.(1- i) databus1.i databus2.i resultbus.i cout.i))
(connect cout cout.n)
)
```

Instead of using the connect statement one could have conditionalized the calculation of the arguments to alu:

```
(macro ALU (n databus1 databus2 resultbus cin cout)
(repeat i 1 n
(alu (cond ((== i 1) cin) (t cout.(1- i)))
     databus1.i
     databus2.i
     resultbus.i
     (cond ((== i n) cout) (t cout.n))) ))
)
```

The file /usr/vlsi/nl/pads.net contains the following macros:

```
(input-pad world)                  ; the input pad
(output-pad world in)               ; the output pad
(tristate-pad world in direction)    ; the tristate pad
(clockbar-pad world ~phi1 ~phi2)      ; the clock pad
```

Changes for NET/PRESIM/RNL from issue of 3.0 documents

Christopher Terman

M.I.T. Laboratory for Computer Science
545 Technology Square, Room 418
Cambridge, MA 02135
9/82
(UW/NW VLSI 6/21/83)

UW/NW VLSI NOTE: The program release for 7-1-83 is version 3.3

Version 3.3 (5/19/83)

1) The switch-level calculation has been modified so that circuits that involve depletion and X devices as pass gates are more accurately handled. For example:

```
e a b gnd
d b b vdd
e x b c
c c 1.0
```

If A=1, and X=1, C = 0 as one would expect. When X->X, C now = 0 instead of X as before.

2) Ratio errors are now reported when the event is processed rather than when the event is created. This means ratio errors that are short-lived compared to the time constant of the node won't get reported at all (ie, many of the ratio errors that used to be reported for CMOS designs won't be in this version). Hopefully, the remaining errors are the significant ones.

3) initialization no longer looks at resistor devices (resistors and two-terminal low-power depletion devices). It shouldn't have in the first place...

4) Fixed resistance table in nlstep to correspond to new transistor type numbers (should have been changed before! CMOS doesn't work in switch mode with the previous arrangement of the table).

5) The resistance calculation done by simstep when calculating a node's final logic state has been modified to fix the unrealistically low estimates caused by X's in the network. This should make things like latches with resistive feedbacks work correctly without fudging the calibration.

6) a new subr (node-time n) has been added that returns the simulated time at which node n made its latest transition; the time is in the same units as current-time. If the node hasn't made any transitions, the reported time is -1.

7) (node-value 'nil) no longer makes rnl loop forever

8) (node foo.bar) no longer makes net loop forever

9) -n flag to net disables error report about use of undeclared name.

10) sim-step, etc. no longer get a memory fault if invoked with no network loaded.

11) Storage allocator in presim fixed so that requests for very large blocks (as might be caused by processing gates with more than 255 inputs!) are handled correctly.

12) Macros for accessing lisp data structures have been added to the code; nothing works differently but the source should be much more portable.

13) VAX/VMS and UNIX: added user interrupt feature: typing the quit character (^C on vmx, ^B on unix) will cause the value of the symbol user-interrupt to be set to t. Programs should reset the user-interrupt to nil if they expect to detect subsequent interrupts.

14) The routines sim-step, etc. no longer get a memory fault if invoked with no network loaded.

Version 3.2 (11/12/82)

1) Switch-level simulation is now part of RNL, rather than a separate program: two new subrs have been added

(switch-step stop-time node-list)

just like sim-step, except transistors are modelled as switches and transitions have unit delay. This algorithm is somewhat faster than the usual RNL calculation for many circuits, but can give X answers for circuits for which transistor size is important for correct operation (e.g., bit line in a dynamic memory). To ensure correct operation, one should not use sim-step until the event list is empty (and vice versa) -- i.e., all events scheduled by a particular algorithm must be handled by the same algorithm; the value of event-list-empty can be tested to see if the all events have been handled.

(switch-init)

like sim-init except uses switch-level simulation algorithm.

These routines may be useful when debugging the basic functionality of a circuit, or when simulating a circuit which has not been correctly sized (one that gives ratio errors using sim-step). Since the switch-level algorithms are much faster when dealing with large groups of interconnected nodes, they may be particularly useful when initializing a network.

2) The read-network subr now does the right thing when called more than once: nodes with the same name in different input files will now refer to the same node internally with the appropriate merging of information from each input file.

3) presim now uses a fourth characteristic resistance, called "power", when doing the power calculation (see the -p option). You can specify values for this resistance in the parameter file like other resistances, e.g.:

resistance pullup power 10 10 1E4

4) current simulation time (stored as value of symbol current-time) is now kept as a floating point number, allowing more time to pass before wrapping around. Value is still in 10th nanoseconds. The step function in nil has to be changed since the % operator doesn't work on flonums: just dividing current-time by 10 before printing it out works just fine (i.e., don't need % operator to get tenths digit).

5) Fixed a number of minor bugs having to do with object stack management on errors. Also fixed a bug in the read routine having to do with dotted pairs.

(All thanks to Doug Brown at Metheus).

Version 3.1 (9/29/82)

1) Added match-node subr:

(match-node pattern)

returns a list of nodes whose names match the pattern; "*" wildcards are allowed.

2) Added resistor as explicit network component. There is a new NET command

(resistor node1 node2 resistance)

that sticks a resistor of the specified size between node1 and node2. PRESIM and RNL know how deal with the results...

3) Two terminal depletion devices (DEP and LOWP) are converted to resistors with a value equal to the transistors' dynamic high resistance. This should improve performance of the circuits in the presence of X's since resistors don't have gates.

4) Corrected anomolous behavior involving nodes with user-specified delays. Basically, if a node A is connected to node B and B has a user-specified delay, B's delay will only contribute to A's delay if B some how affects A's value (i.e., if B has some drive associated with it).

5) Made invocation of the charge sharing calculation more selective: in theory calculation needs only be done when nodes are added to the network. Changed code so that connecting an input to a network (such as a p-channel pullup) does not invoke charge sharing. Should speed up CMOS calculations.

6) The charge sharing calculation is now controlled by maxres parameter: the calculation will not look beyond transistors with a static resistance that is greater than or equal to maxres.

7) Added log-file subr that allows the user to keep a log of an RNL session. The file will contain all user type-in as well as copies of output to the stdout and stderr streams.

8) Added write-ascii-state subr which writes a readable file telling the state of each node in the network.

9) A -p option was added to presim:

-pvoltage

provides a worse-case estimate of the circuit power consumption by assuming that all the pullups (DEP or LOWP devices with drain=VDD) are all on simultaneously. "voltage" specifies the supply voltage, for example "-p5" specifies a VDD of 5 volts. The result is printed after PRESIM completes its other processing.

10) A new format .sim record was added that allows node extractors to specify areas and perimeters for 2nd metal, 1st metal, poly, diffusion, and p-diffusion; the corresponding capacitance parameters were added to presim. See sim.doc and presim.doc for details.

Version 3.0 (9/11/82)

1) Initial documented release, start of this file.

User's Guide to M.I.T. ".sim" File Format

Christopher Terman

M.I.T. Laboratory for Computer Science
545 Technology Square, Room 418
Cambridge, MA 02135

1.0 Introduction

The .sim files are ASCII files used by various programs to describe MOS transistor networks and their associated parameters. The network extractor and NETLIST program output .sim files; an electrical rules checking program (stat) and PRESIM read the files.

(UW/NW VLSI NOTE: The .sim file used here is incompatible with the .sim used at U. Cal. Berkeley. See mextra documentation for UCB .sim file format)

2.0 Record Types

Each line of a .sim file is a separate "record" whose type is determined by the first character of the line. The possible record types are:

```
| ...
```

Lines beginning with vertical bar (|) are treated as comments and ignored by programs that read .sim files.

```
@ filename
```

The "@ filename" command redirects input to the name file. When the end-of-file is reached, input reverts to the current file at the following line. Indirect files can be nested; usually there is some system dependent limit on the number of simultaneously open files which limits the depth of nesting.

```
e gate source drain length width [rpa] xpos ypos area
i gate source drain length width [rpa] xpos ypos area
p gate source drain length width [rpa] xpos ypos area
d gate source drain length width [rpa] xpos ypos area
l gate source drain length width [rpa] xpos ypos area
```

Above are the possible transistor records. The first character tells the type of the transistor:

e	n-channel enhancement
i	n-channel zero threshold (intrinsic) enhancement

p p-channel enhancement

d depletion

l low-power depletion

The next three parameters are the names of the gate, source, and drain nodes. The length and width can be either integers or floating-point giving the dimensions of the active transistor area; the units are up to the user who should set the correct conversion factor to microns in PRESIM using the LAMBDA parameter.

The next parameter is a letter specifying the shape of the transistor:

r rectangular

p rectangular, monotonically increasing width

a other shapes...

NETLIST always puts an "r" in this field; the network extractor uses the other letters to tell what approximation was used to derive an effective length width from area and perimeter info when the transistor is non-rectangular.

"xpos" and "ypos" tell the coordinates of the upper left hand corner of the transistor (NETLIST always specifies 0,0).

"area" tells the true active area; may be different from width*length if the network extractor used an approximation (in output from NETLIST, area always equal width*length).

N node xpos ypos M-area P-area D-area D-perim

Node record from the network extractor. Tells the node name, coordinates of the upper left corner of the node and various geometrical info:

M-area metal area

P-area polysilicon area (including over trans's).

D-area diffusion area

D-perim diffusion perimeter (including next to trans's). The perimeter measure is half of the actual total perimeter (i.e., it is the sum of the lengths of the top and one side).

PRESIM uses these numbers to compute interconnect capacitances from layout information; NETLIST never produces this type of record.

M node xpos ypos M2A M2P MA MP PA PP DA DP PDA PDP

This record is similar to (N) in a new format that includes new information. Tells the node name, coordinates of the upper left corner of the node and various geometrical info:

M2A area of 2nd-level metal

M2P perimeter of 2nd-level metal

MA area of 1st-level metal

MP perimeter of 1st-level metal

PA area of polysilicon

PP perimeter of polysilicon

DA area of n-diffusion

DP perimeter of n-diffusion

DA area of p-diffusion

PDP perimeter of p-diffusion

The perimeter measures are half of the actual total perimeter (i.e., they are the sum of the lengths of the top and one side). PRESIM uses these numbers to compute interconnect capacitances from layout information; NETLIST never produces this type of record.

c node cap

Used to specify node capacitance in pf. cap can be either an integer or floating-point number. This is the type of record used by NETLIST to described user-specified capacitors.

= node1 node2 ...

Used to establish a name equivalence. Indicates that two or more names refer to the same electrical node. Parameters for the specified nodes are merged; the user can subsequently use the names interchangeably.

D node low-to-high high-to-low

Describes user-specified delays for the given node. Delays are integral numbers of RNL time units (1/10th nanosecond).

t node low high

User specified node thresold. Describes user-specified voltage thresholds in normalized voltage units (i.e., low and high are floating-point numbers in the range 0.0 to 1.0).

SPICE User's Guide

A.Vladimirescu, Kaihe Zhang,
A.R.Newton, D.O.Pederson, A.Sangiovanni-Vincentelli

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, Ca., 94720

This manual corresponds to SPICE version 2G6

Acknowledgement: Dr. Richard Dowell and Dr. Sally Liu have contributed to develop the present SPICE version. SPICE was originally developed by Dr. Lawrence Nagel and has been modified extensively by Dr. Ellis Cohen.

SPICE is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analyses. Circuits may contain resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four types of dependent sources, transmission lines, and the four most common semiconductor devices: diodes, BJT's, JFET's, and MOSFET's.

SPICE has built-in models for the semiconductor devices, and the user need specify only the pertinent model parameter values. The model for the BJT is based on the integral charge model of Gummel and Poon; however, if the Gummel- Poon parameters are not specified, the model reduces to the simpler Ebers-Moll model. In either case, charge storage effects, ohmic resistances, and a current-dependent output conductance may be included. The diode model can be used for either junction diodes or Schottky barrier diodes. The JFET model is based on the FET model of Shichman and Hodges. Three MOSFET models are implemented; MOS1 is described by a square-law I-V characteristic MOS2 is an analytical model while MOS3 is a semi-empirical model. Both MOS2 and MOS3 include second-order effects such as channel length modulation, subthreshold conduction, scattering limited velocity saturation, small size effects and charge-controlled capacitances.

1. TYPES OF ANALYSIS

1.1. DC Analysis

The dc analysis portion of SPICE determines the dc operating point of the circuit with inductors shorted and capacitors opened. A dc analysis is automatically performed prior to a transient analysis to determine the transient initial conditions, and prior to an ac small-signal analysis to determine the linearized, small-signal models for nonlinear devices. If requested, the dc small-signal value of a transfer function (ratio of output variable to input

source), input resistance, and output resistance will also be computed as a part of the dc solution. The dc analysis can also be used to generate dc transfer curves: a specified independent voltage or current source is stepped over a user-specified range and the dc output variables are stored for each sequential source value. If requested, SPICE also will determine the dc small-signal sensitivities of specified output variables with respect to circuit parameters. The dc analysis options are specified on the .DC, .TF, .OP, and .SENS control cards.

If one desires to see the small signal models for nonlinear devices in conjunction with a transient analysis operating point, then the .OP card must be provided. The dc bias conditions will be identical for each case, but the more comprehensive operating point information is not available to be printed when transient initial conditions are computed.

1.2. AC Small-Signal Analysis

The ac small-signal portion of SPICE computes the ac output variables as a function of frequency. The program first computes the dc operating point of the circuit and determines linearized, small-signal models for all of the nonlinear devices in the circuit. The resultant linear circuit is then analyzed over a user-specified range of frequencies. The desired output of an ac small- signal analysis is usually a transfer function (voltage gain, transimpedance, etc). If the circuit has only one ac input, it is convenient to set that input to unity and zero phase, so that output variables have the same value as the transfer function of the output variable with respect to the input.

The generation of white noise by resistors and semiconductor devices can also be simulated with the ac small-signal portion of SPICE. Equivalent noise source values are determined automatically from the small-signal operating point of the circuit, and the contribution of each noise source is added at a given summing point. The total output noise level and the equivalent input noise level are determined at each frequency point. The output and input noise levels are normalized with respect to the square root of the noise bandwidth and have the units Volts/rt Hz or Amps/rt Hz. The output noise and equivalent input noise can be printed or plotted in the same fashion as other output variables. No additional input data are necessary for this analysis.

Flicker noise sources can be simulated in the noise analysis by including values for the parameters KF and AF on the appropriate device model cards.

The distortion characteristics of a circuit in the small signal mode can be simulated as a part of the ac small-signal analysis. The analysis is performed assuming that one or two signal frequencies are imposed at the input.

The frequency range and the noise and distortion analysis parameters are specified on the .AC, .NOISE, and .DISTO control lines.

1.3. Transient Analysis

The transient analysis portion of SPICE computes the transient output variables as a function of time over a user specified time interval. The initial conditions are automatically determined by a dc analysis. All sources which are not time dependent (for example, power supplies) are set to their dc value. For large-signal sinusoidal simulations, a Fourier analysis of the output waveform can be specified to obtain the frequency domain Fourier coefficients. The transient time interval and the Fourier analysis options are specified on the .TRAN and .FOURIER control lines.

1.4. Analysis at Different Temperatures

All input data for SPICE is assumed to have been measured at 27 deg C (300 deg K). The simulation also assumes a nominal temperature of 27 deg C. The circuit can be simulated at other temperatures by using a .TEMP control line.

Temperature appears explicitly in the exponential terms of the BJT and diode model equations. In addition, saturation currents have a built-in temperature dependence. The

temperature dependence of the saturation current in the BJT models is determined by:

$$IS(T1)=IS(T0)[(\frac{T1}{T0})^{XTI} e^{\frac{qEG(T1-T0)}{k\cdot T1\cdot T0}}]$$

where k is Boltzmann's constant, q is the electronic charge, EG is the energy gap which is a model parameter, and XTI is the saturation current temperature exponent (also a model parameter, and usually equal to 3). The temperature dependence of forward and reverse beta is according to the formula:

$$beta(T1)=beta(T0)[(\frac{T1}{T0})^{XTB}]$$

where T1 and T0 are in degrees Kelvin, and XTB is a user-supplied model parameter. Temperature effects on beta are carried out by appropriate adjustment to the values of BF, ISE, BR, and ISC. Temperature dependence of the saturation current in the junction diode model is determined by:

$$IS(T1)=IS(T0)[(\frac{T1}{T0})^{(\frac{XTI}{N})} \frac{e^{qEG(T1-T0)}}{(kN\cdot T1\cdot T0)}]$$

where N is the emission coefficient, which is a model parameter, and the other symbols have the same meaning as above. Note that for Schottky barrier diodes, the value of the saturation current temperature exponent, XTI, is usually 2.

Temperature appears explicitly in the value of junction potential, PHI, for all the device models. The temperature dependence is determined by:

$$PHI(TEMP)=\frac{kTEMP}{q\log(\frac{NaNd}{Ni(TEMP)^2})}$$

where k is Boltzmann's constant, q is the electronic charge, Na is the acceptor impurity density, Nd is the donor impurity den sity, Ni is the intrinsic concentration, and EG is the energy gap.

Temperature appears explicitly in the value of surface mobility, UO, for the MOSFET model. The temperature dependence is determined by:

$$UO(TEMP)=\frac{UO(TNOM)}{(TEMP/TNOM)^{1.5}}$$

The effects of temperature on resistors is modeled by the formula:

$$value(TEMP)=value(TNOM)[1+TC1(TEMP-TNOM)+TC2[(TEMP-TNOM)^2]$$

where TEMP is the circuit temperature, TNOM is the nominal temperature, and TC1 and TC2 are the first- and second-order temperature coefficients.

2. CONVERGENCE

Both dc and transient solutions are obtained by an iterative process which is terminated when both of the following conditions hold:

1) The nonlinear branch currents converge to within a tolerance of 0.1 percent or 1 picoamp (1.0E-12 Amp), whichever is larger.
2) The node voltages converge to within a tolerance of 0.1 per cent or 1 microvolt (1.0E-6 Volt), whichever is larger.

Although the algorithm used in SPICE has been found to be very reliable, in some cases it will fail to converge to a solution. When this failure occurs, the program will print the node voltages at the last iteration and terminate the job. In such cases, the node voltages that are printed are not necessarily correct or even close to the correct solution.

Failure to converge in the dc analysis is usually due to an error in specifying circuit connections, element values, or model parameter values. Regenerative switching circuits or circuits with positive feedback probably will not converge in the dc analysis unless the OFF option is used for some of the devices in the feedback path, or the .NODESET card is used to force the circuit to converge to the desired state.

3. INPUT FORMAT

The input format for SPICE is of the free format type. Fields on a card are separated by one or more blanks, a comma, an equal (=) sign, or a left or right parenthesis; extra spaces are ignored. A card may be continued by entering a + (plus) in column 1 of the following card; SPICE continues reading beginning with column 2.

A name field must begin with a letter (A through Z) and cannot contain any delimiters. Only the first eight characters of the name are used.

A number field may be an integer field (12, -44), a floating point field (3.14159), either an integer or floating point number followed by an integer exponent (1E-14, 2.65E3), or either an integer or a floating point number followed by one of the following scale factors:

T=1E12 G=1E9 MEG=1E6 K=1E3 MIL=25.4E-6 M=1E-3 U=1E-6 N=1E-9
P=1E-12 F=1E-15

Letters immediately following a number that are not scale factors are ignored, and letters immediately following a scale factor are ignored. Hence, 10, 10V, 10VOLTS, and 10HZ all represent the same number, and M, MA, MSEC, and MMHOS all represent the same scale factor. Note that 1000, 1000.0, 1000HZ, 1E3, 1.0E3, 1KHZ, and 1K all represent the same number.

4. CIRCUIT DESCRIPTION

The circuit to be analyzed is described to SPICE by a set of element cards, which define the circuit topology and element values, and a set of control cards, which define the model parameters and the run controls. The first card in the input deck must be a title card, and the last card must be a .END card. The order of the remaining cards is arbitrary (except, of course, that continuation cards must immediately follow the card being continued).

Each element in the circuit is specified by an element card that contains the element name, the circuit nodes to which the element is connected, and the values of the parameters that determine the electrical characteristics of the element. The first letter of the element name specifies the element type. The format for the SPICE element types is given in what follows. The strings XXXXXXX, YYYYYYY, and ZZZZZZZ denote arbitrary alphanumeric strings. For example, a resistor name must begin with the letter R and can contain from one to eight characters. Hence, R, R1, RSE, ROUT, and R3AC2ZY are valid resistor names.

Data fields that are enclosed in lt and gt signs '< >' are optional. All indicated punctuation (parentheses, equal signs, etc.) are required. With respect to branch voltages and currents, SPICE uniformly uses the associated reference convention (current flows in the direction of voltage drop).

Nodes must be nonnegative integers but need not be numbered sequentially. The datum (ground) node must be numbered zero. The circuit cannot contain a loop of voltage sources and/or inductors and cannot contain a cutset of current sources and/or capacitors. Each node in the circuit must have a dc path to ground. Every node must have at least two connections except for transmission line nodes (to permit unterminated transmission lines) and MOSFET substrate nodes (which have two internal connections anyway).

5. TITLE CARD, COMMENT CARDS AND .END CARD

5.1. Title Card

Examples:

```
POWER AMPLIFIER CIRCUIT
TEST OF CAM CELL
```

This card must be the first card in the input deck. Its contents are printed verbatim as the heading for each section of output.

5.2. .END Card

Examples:

```
.END
```

This card must always be the last card in the input deck. Note that the period is an integral part of the name.

5.3. Comment Card

General Form:

```
* < any comment >
```

Examples:

```
* RF=1K      GAIN SHOULD BE 100
* MAY THE FORCE BE WITH MY CIRCUIT
```

The asterisk in the first column indicates that this card is a comment card. Comment cards may be placed anywhere in the circuit description.

6. ELEMENT CARDS

6.1. Resistors

General form:

```
RXXXXXXX N1 N2 VALUE < TC=TC1<,TC2> >
```

Examples:

```
R1 1 2 100
RC1 12 17 1K TC=0.001,0.015
```

N1 and N2 are the two element nodes. VALUE is the resistance (in ohms) and may be positive or negative but not zero. TC1 and TC2 are the (optional) temperature coefficients; if not specified, zero is assumed for both. The value of the resistor as a function of temperature is given by:

$$value(TEMP) = value(TNOM)[1 + TC1(TEMP - TNOM) + TC2[(TEMP - TNOM)^2]$$

6.2. Capacitors and Inductors

General form:

CXXXXXXX N+ N- VALUE <IC=INCOND>
LYYYYYYY N+ N- VALUE <IC=INCOND>

Examples:

CBYP 13 0 1UF
COSC 17 23 10U IC=3V
LLINK 42 69 1UH
LSHUNT 23 51 10U IC=15.7MA

N+ and N- are the positive and negative element nodes, respectively. VALUE is the capacitance in Farads or the inductance in Henries.

For the capacitor, the (optional) initial condition is the initial (time-zero) value of capacitor voltage (in Volts). For the inductor, the (optional) initial condition is the initial (time-zero) value of inductor current (in Amps) that flows from N+, through the inductor, to N-. Note that the initial conditions (if any) apply 'only' if the UIC option is specified on the .TRAN card.

Nonlinear capacitors and inductors can be described.

General form :

CXXXXXXX N+ N- POLY C0 C1 C2 ... <IC=INCOND>
LYYYYYYY N+ N- POLY L0 L1 L2 ... <IC=INCOND>

C0 C1 C2 ...(and L0 L1 L2 ...) are the coefficients of a polynomial describing the element value. The capacitance is expressed as a function of the voltage across the element while the inductance is a function of the current through the inductor. The value is computed as

$$value = C0 + C1 \cdot V + C2 \cdot V^2 + \cdots$$

$$value = L0 + L1 \cdot I + L2 \cdot I^2 + \cdots$$

where V is the voltage across the capacitor and I the current flowing in the inductor.

6.3. Coupled (Mutual) Inductors

General form:

KXXXXXXX LYYYYYYY LZZZZZZZ VALUE

Examples:

K43 LAA LBB 0.999
KXFRMR L1 L2 0.87

LYYYYYYY and LZZZZZZZ are the names of the two coupled inductors, and VALUE is the coefficient of coupling, K, which must be greater than 0 and less than or equal to 1. Using the 'dot' convention, place a 'dot' on the first node of each inductor.

6.4. Transmission Lines (Lossless)

General form:

```
TXXXXXXX N1 N2 N3 N4 Z0=VALUE <TD=VALUE> <F=FREQ <NL=NRMLEN>>
+                <IC=V1,I1,V2,I2>
```

Examples:

```
T1 1 0 2 0 Z0=50 TD=10NS
```

N1 and N2 are the nodes at port 1; N3 and N4 are the nodes at port 2. Z0 is the characteristic impedance. The length of the line may be expressed in either of two forms. The transmission delay, TD, may be specified directly (as TD=10ns, for example). Alternatively, a frequency F may be given, together with NL, the normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency F. If a frequency is specified but NL is omitted, 0.25 is assumed (that is, the frequency is assumed to be the quarter-wave frequency). Note that although both forms for expressing the line length are indicated as optional, one of the two must be speci fied.

Note that this element models only one propagating mode. If all four nodes are distinct in the actual circuit, then two modes may be excited. To simulate such a situation, two transmission line elements are required. (see the example in Appendix A for further clarification.)

The (optional) initial condition specification consists of the voltage and current at each of the transmission line ports. Note that the initial conditions (if any) apply 'only' if the UIC option is specified on the .TRAN card.

One should be aware that SPICE will use a transient time step which does not exceed 1/2 the minimum transmission line delay. Therefore very short transmission lines (compared with the analysis time frame) will cause long run times.

6.5. Linear Dependent Sources

SPICE allows circuits to contain linear dependent sources characterized by any of the four equations

$$i = g \cdot v \qquad v = e \cdot v \qquad i = f \cdot i \qquad v = h \cdot i$$

where g, e, f, and h are constants representing transconductance, voltage gain, current gain, and transresistance, respectively. Note: a more complete description of dependent sources as imple mented in SPICE is given in Appendix B.

6.6. Linear Voltage-Controlled Current Sources

General form:

```
GXXXXXXX N+ N- NC+ NC- VALUE
```

Examples:

```
G1 2 0 5 0 0.1MMHO
```

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. NC+ and NC- are the positive and negative controlling nodes, respectively. VALUE is the transconductance (in mhos).

6.7. Linear Voltage-Controlled Voltage Sources

General form:

EXXXXXXX N+ N- NC+ NC- VALUE

Examples:

E1 2 3 14 1 2.0

N+ is the positive node, and N- is the negative node. NC+ and NC- are the positive and negative controlling nodes, respectively. VALUE is the voltage gain.

6.8. Linear Current-Controlled Current Sources

General form:

FXXXXXXX N+ N- VNAM VALUE

Examples:

F1 13 5 VSENS 5

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. VNAM is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of VNAM. VALUE is the current gain.

6.9. Linear Current-Controlled Voltage Sources

General form:

HXXXXXXX N+ N- VNAM VALUE

Examples:

HX 5 17 VZ 0.5K

N+ and N- are the positive and negative nodes, respectively. VNAM is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of VNAM. VALUE is the transresistance (in ohms).

6.10. Independent Sources

General form:

VXXXXXXX N+ N- <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
IYYYYYYY N+ N- <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>

Examples:

VCC 10 0 DC 6

```
VIN 13 2 0.001 AC 1 SIN(0 1 1MEG)
ISRC 23 21 AC 0.333 45.0 SFFM(0 1 10K 5 1K)
VMEAS 12 9
```

N+ and N- are the positive and negative nodes, respectively. Note that voltage sources need not be grounded. Positive current is assumed to flow from the positive node, through the source, to the negative node. A current source of positive value, will force current to flow out of the N+ node, through the source, and into the N- node. Voltage sources, in addition to being used for circuit excitation, are the 'ammeters' for SPICE, that is, zero valued voltage sources may be inserted into the circuit for the purpose of measuring current. They will, of course, have no effect on circuit operation since they represent short-circuits.

DC/TRAN is the dc and transient analysis value of the source. If the source value is zero both for dc and transient analyses, this value may be omitted. If the source value is time-invariant (e.g., a power supply), then the value may optionally be preceded by the letters DC.

ACMAG is the ac magnitude and ACPHASE is the ac phase. The source is set to this value in the ac analysis. If ACMAG is omitted following the keyword AC, a value of unity is assumed. If ACPHASE is omitted, a value of zero is assumed. If the source is not an ac small-signal input, the keyword AC and the ac values are omitted.

Any independent source can be assigned a time-dependent value for transient analysis. If a source is assigned a time- dependent value, the time-zero value is used for dc analysis. There are five independent source functions: pulse, exponential, sinusoidal, piecewise linear, and single-frequency FM. If parameters other than source values are omitted or set to zero, the default values shown will be assumed. (TSTEP is the printing increment and TSTOP is the final time (see the .TRAN card for explanation)).

1. Pulse PULSE(V1 V2 TD TR TF PW PER)

Examples:

```
VIN 3 0 PULSE(-1 1 2NS 2NS 2NS 50NS 100NS)
```

parameter	default	units
V1 (initial value)		Volts or Amps
V2 (pulsed value)		Volts or Amps
TD (delay time)	0.0	seconds
TR (rise time)	TSTEP	seconds
TF (fall time)	TSTEP	seconds
PW (pulse width)	TSTOP	seconds
PER(period)	TSTOP	seconds

A single pulse so specified is described by the following table:

time	value
0	V1
TD	V1
TD+TR	V2
TD+TR+PW	V2
TD+TR+PW+TF	V1
TSTOP	V1

Intermediate points are determined by linear interpolation.

2. Sinusoidal SIN(VO VA FREQ TD THETA)

Examples:

VIN 3 0 SIN(0 1 100MEG 1NS 1E10)

parameter	default value	units
VO	(offset)	Volts or Amps
VA	(amplitude)	Volts or Amps
FREQ	(frequency)	1/TSTOP
TD	(delay)	0.0
THETA	(damping factor)	0.0

The shape of the waveform is described by the following table:

time	value
0 to TD	VO
TD to TSTOP	$V0+VAe^{(-(time-TD)\Theta)}\sin(2\pi FREQ(time+TD))$

3. Exponential EXP(V1 V2 TD1 TAU1 TD2 TAU2)

Examples:

VIN 3 0 EXP(-4 -1 2NS 30NS 60NS 40NS)

parameters	default values	units
V1	(initial value)	Volts or Amps
V2	(pulsed value)	Volts or Amps
TD1	(rise delay time)	0.0
TAU1	(rise time constant)	TSTEP
TD2	(fall delay time)	TD1+TSTEP
TAU2	(fall time constant)	TSTEP

The shape of the waveform is described by the following table:

time	value
0 to TD1	V1
TD1 to TD2	$V1+(V2-V1)[1-e^{(-(time-TD1)/TAU1)}]$
TD2 to TSTOP	$V1+(V2-V1)[1-e^{(-(time-TD1)/TAU1)}+(V1-V2)[1-e^{(-(time-TD2)/TAU2)}]$

4. Piece-Wise Linear PWL(T1 V1 < T2 V2 T3 V3 T4 V4 ...>)

Example:

VCLOCK 7 5 PWL(0 -7 10NS -7 11NS -3 17NS -3 18NS -7 50NS -7)

Each pair of values (Ti, Vi) specifies that the value of the source is Vi (in Volts or Amps) at time=Ti. The value of the source at intermediate values of time is determined by using linear interpolation on the input values.

5. Single-Frequency FM SFFM(VO VA FC MDI FS)

Examples:

V1 12 0 SFFM(0 1M 20K 5 1K)

parameters	default values	units
VO	(offset)	Volts or Amps
VA	(amplitude)	Volts or Amps
FC	(carrier frequency)	1/TSTOP
MDI	(modulation index)	
FS	(signal frequency)	1/TSTOP

The shape of the waveform is described by the following equation:

$$value + VO + VA\sin((2\pi \cdot FC \cdot time) + MDI\sin(2\pi \cdot FS \cdot time))$$

7. SEMICONDUCTOR DEVICES

The elements that have been described to this point typically require only a few parameter values to specify completely the electrical characteristics of the element. However, the models for the four semiconductor devices that are included in the SPICE program require many parameter values. Moreover, many devices in a circuit often are defined by the same set of device model parameters. For these reasons, a set of device model parameters is defined on a separate .MODEL card and assigned a unique model name. The device element cards in SPICE then reference the model name. This scheme alleviates the need to specify all of the model parameters on each device element card.

Each device element card contains the device name, the nodes to which the device is connected, and the device model name. In addition, other optional parameters may be specified for each device: geometric factors and an initial condition.

The area factor used on the diode, BJT and JFET device card determines the number of equivalent parallel devices of a specified model. The affected parameters are marked with an asterisk under the heading 'area' in the model descriptions below. Several geometric factors associated with the channel and the drain and source diffusions can be specified on the MOSFET device card.

Two different forms of initial conditions may be specified for devices. The first form is included to improve the dc convergence for circuits that contain more than one stable state. If a device is specified OFF, the dc operating point is deter mined with the terminal voltages for that device set to zero. After convergence is obtained, the program continues to iterate to obtain the exact value for the terminal voltages. If a circuit has more than one dc stable state, the OFF option can be used to force the solution to correspond to a desired state. If a device is specified OFF when in reality the device is conducting, the program will still obtain the correct solution (assuming the solutions converge) but more iterations will be required since the program must independently converge to two separate solutions. The .NODESET card serves a similar purpose as the OFF option. The .NODESET

option is easier to apply and is the preferred means to aid convergence.

The second form of initial conditions are specified for use with the transient analysis. These are true 'initial conditions' as opposed to the convergence aids above. See the description of the .IC card and the .TRAN card for a detailed explanation of initial conditions.

7.1. Junction Diodes

General form:

DXXXXXXX N+ N- MNAME <AREA> <OFF> <IC=VD>

Examples:

```
DBRIDGE 2 10 DIODE1
DCLMP 3 7 DMOD 3.0 IC=0.2
```

N+ and N- are the positive and negative nodes, respectively. MNAME is the model name, AREA is the area factor, and off indicates an (optional) starting condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification using IC=VD is intended for use with the UIC option on the .TRAN card, when a transient analysis is desired starting from other than the quiescent operating point.

7.2. Bipolar Junction Transistors (BJT's)

General form:

QXXXXXXX NC NB NE <NS> MNAME <AREA> <OFF> <IC=VBE,VCE>

Examples:

```
Q23 10 24 13 QMOD IC=0.6,5.0
Q50A 11 26 4 20 MOD1
```

NC, NB, and NE are the collector, base, and emitter nodes, respectively. NS is the (optional) substrate node. If unspecified, ground is used. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) initial condition on the device for the dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification using IC=VBE,VCE is intended for use with the UIC option on the .TRAN card, when a transient analysis is desired starting from other than the quiescent operating point. See the .IC card description for a better way to set transient initial conditions.

7.3. Junction Field-Effect Transistors (JFET's)

General form:

JXXXXXXX ND NG NS MNAME <AREA> <OFF> <IC=VDS,VGS>

Examples:

```
J1 7 2 3 JM1 OFF
```

ND, NG, and NS are the drain, gate, and source nodes, respectively. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) initial condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification, using IC=VDS,VGS is intended for use with the UIC option on the .TRAN card, when a transient analysis is desired starting from other than the quiescent operating point (see the .IC card for a better way to set initial conditions).

7.4. MOSFET's

General form:

```
MXXXXXXX ND NG NS NB MNAME <L=VAL> <W=VAL> <AD=VAL>
<AS=VAL>
+ <PD=VAL> <PS=VAL> <NRD=VAL> <NRS=VAL> <OFF>
<IC=VDS,VGS,VBS>
```

Examples:

```
M1 24 2 0 20 TYPE1
M31 2 17 6 10 MODM L=5U W=2U
M31 2 16 6 10 MODM 5U 2U
M1 2 9 3 0 MOD1 L=10U W=5U AD=100P AS=100P PD=40U PS=40U
M1 2 9 3 0 MOD1 10U 5U 2P 2P
```

ND, NG, NS, and NB are the drain, gate, source, and bulk (substrate) nodes, respectively. MNAME is the model name. L and W are the channel length and width, in meters. AD and AS are the areas of the drain and source diffusions, in sq-meters. Note that the suffix U specifies microns (1E-6 m) and P sq-microns (1E-12 sq-m). If any of L, W, AD, or AS are not specified, default values are used. The user may specify the values to be used for these default parameters on the .OPTIONS card. The use of defaults simplifies input deck preparation, as well as the editing required if device geometries are to be changed. PD and PS are the perimeters of the drain and source junctions, in meters. NRD and NRS designate the equivalent number of squares of the drain and source diffusions; these values multiply the sheet resistance RSH specified on the .MODEL card for an accurate representation of the parasitic series drain and source resistance of each transistor. PD and PS default to 0.0 while NRD and NRS to 1.0. OFF indicates an (optional) initial condition on the device for dc analysis. The (optional) initial condition specification using IC=VDS,VGS,VBS is intended for use with the UIC option on the .TRAN card, when a transient analysis is desired starting from other than the quiescent operating point. See the .IC card for a better and more convenient way to specify transient initial conditions.

7.5. .MODEL Card

General form:

```
.MODEL MNAME TYPE(PNAME1=PVAL1 PNAME2=PVAL2 ... )
```

Examples:

```
.MODEL MOD1 NPN BF=50 IS=1E-13 VBF=50
```

The .MODEL card specifies a set of model parameters that will be used by one or more devices. MNAME is the model name, and type is one of the following seven types:

type	description

NPN	NPN BJT model
PNP	PNP BJT model
D	diode model
NJF	N-channel JFET model
PJF	P-channel JFET model
NMOS	N-channel MOSFET model
PMOS	P-channel MOSFET model

Parameter values are defined by appending the parameter name, as given below for each model type, followed by an equal sign and the parameter value. Model parameters that are not given a value are assigned the default values given below for each model type.

7.6. Diode Model

The dc characteristics of the diode are determined by the parameters IS and N. An ohmic resistance, RS, is included. Charge storage effects are modeled by a transit time, TT, and a nonlinear depletion layer capacitance which is determined by the parameters CJO, VJ, and M. The temperature dependence of the saturation current is defined by the parameters EG, the energy and XTI, the saturation current temperature exponent. Reverse breakdown is modeled by an exponential increase in the reverse diode current and is determined by the parameters BV and IBV (both of which are positive numbers).

	name	parameter	units	default	example	area
1	IS	saturation current	A	1.0E-14	1.0E-14	*
2	RS	ohmic resistance	Ohm	0	10	*
3	N	emission coefficient	-	1	1.0	
4	TT	transit-time	sec	0	0.1Ns	
5	CJO	zero-bias junction capacitance	F	0	2PF	*
6	VJ	junction potential	V	1	0.6	
7	M	grading coefficient	-	0.5	0.5	
8	EG	activation energy	eV	1.11	1.11 Si 0.69 Sbd 0.67 Ge	
9	XTI	saturation-current temp. exp	-	3.0	3.0 jn 2.0 Sbd	
10	KF	flicker noise coefficient	-	0		
11	AF	flicker noise exponent	-	1		
12	FC	coefficient for forward-bias depletion capacitance formula	-	0.5		
13	BV	reverse breakdown voltage	V	infinite	40.0	
14	IBV	current at breakdown voltage	A	1.0E-3		

7.7. BJT Models (both NPN and PNP)

The bipolar junction transistor model in SPICE is an adaptation of the integral charge control model of Gummel and Poon. This modified Gummel-Poon model extends the original model to include several effects at high bias levels. The model will automatically simplify to the simpler Ebers-Moll model when certain parameters are not specified. The parameter names used in the modified Gummel-Poon model have been chosen to be more easily understood by the program user, and to reflect better both physical and circuit design thinking.

The dc model is defined by the parameters IS, BF, NF, ISE, IKF, and NE which determine the forward current gain characteristics, IS, BR, NR, ISC, IKR, and NC which determine the reverse current gain characteristics, and VAF and VAR which determine the

output conductance for forward and reverse regions. Three ohmic resistances RB, RC, and RE are included, where RB can be high current dependent. Base charge storage is modeled by forward and reverse transit times, TF and TR, the forward transit time TF being bias dependent if desired, and nonlinear depletion layer capacitances which are determined by CJE, VJE, and MJE for the B-E junction , CJC, VJC, and MJC for the B-C junction and CJS, VJS, and MJS for the C-S (Collector-Substrate) junction. The temperature dependence of the saturation current, IS, is determined by the energy-gap, EG, and the saturation current temperature exponent, XTI. Additionally base current temperature dependence is modeled by the beta temperature exponent XTB in the new model.

The BJT parameters used in the modified Gummel-Poon model are listed below. The parameter names used in earlier versions of SPICE2 are still accepted.

Modified Gummel-Poon BJT Parameters

	name	parameter	units	default	example	area
1	IS	transport saturation current	A	1.0E-16	1.0E-15	•
2	BF	ideal maximum forward beta	-	100	100	
3	NF	forward current emission coefficient	-	1.0	1	
4	VAF	forward Early voltage	V	infinite	200	
5	IKF	corner for forward beta high current roll-off	A	infinite	0.01	•
6	ISE	B-E leakage saturation current	A	0	1.0E-13	•
7	NE	B-E leakage emission coefficient	-	1.5	2	
8	BR	ideal maximum reverse beta	-	1	0.1	
9	NR	reverse current emission coefficient	-	1	1	
10	VAR	reverse Early voltage	V	infinite	200	
11	IKR	corner for reverse beta high current roll-off	A	infinite	0.01	•
12	ISC	B-C leakage saturation current	A	0	1.0E-13	•
13	NC	B-C leakage emission coefficient	-	2	1.5	
14	RB	zero bias base resistance	Ohms	0	100	•
15	IRB	current where base resistance falls halfway to its min value	A	infinite	0.1	•
16	RBM	minimum base resistance at high currents	Ohms	RB	10	•
17	RE	emitter resistance	Ohms	0	1	•
18	RC	collector resistance	Ohms	0	10	•
19	CJE	B-E zero-bias depletion capacitance	F	0	2PF	•
20	VJE	B-E built-in potential	V	0.75	0.6	
21	MJE	B-E junction exponential factor	-	0.33	0.33	
22	TF	ideal forward transit time	sec	0	0.1Ns	
23	XTF	coefficient for bias dependence of TF	-	0		
24	VTF	voltage describing VBC dependence of TF	V	infinite		
25	ITF	high-current parameter for effect on TF	A	0	•	
26	PTF	excess phase at $freq = 1.0/(TF \cdot 2PI)$	Hz	deg	0	
27	CJC	B-C zero-bias depletion capacitance	F	0	2PF	•
28	VJC	B-C built-in potential	V	0.75	0.5	
29	MJC	B-C junction exponential factor	-	0.33	0.5	
30	XCJC	fraction of B-C depletion capacitance connected to internal base node	-	1		
31	TR	ideal reverse transit time	sec	0	10Ns	
32	CJS	zero-bias collector-substrate				

		capacitance	F	0	2PF	*
33	VJS	substrate junction built-in potential	V	0.75		
34	MJS	substrate junction exponential factor	-	0	0.5	
35	XTB	forward and reverse beta temperature exponent	-	0		
36	EG	energy gap for temperature effect on IS	eV	1.11		
37	XTI	temperature exponent for effect on IS	-	3		
38	KF	flicker-noise coefficient	-	0		
39	AF	flicker-noise exponent	-	1		
40	FC	coefficient for forward-bias depletion capacitance formula	-	0.5		

7.8. JFET Models (both N and P Channel)

The JFET model is derived from the FET model of Shichman and Hodges. The dc characteristics are defined by the parameters VTO and BETA, which determine the variation of drain current with gate voltage, LAMBDA, which determines the output conductance, and IS, the saturation current of the two gate junctions. Two ohmic resistances, RD and RS, are included. Charge storage is modeled by nonlinear depletion layer capacitances for both gate junctions which vary as the -1/2 power of junction voltage and are defined by the parameters CGS, CGD, and PB.

	name	parameter	units	default	example	area
1	VTO	threshold voltage	V	-2.0	-2.0	
2	BETA	transconductance parameter	A/V^2	1.0E-4	1.0E-3	*
3	LAMBDA	channel length modulation parameter	1/V	0	1.0E-4	
4	RD	drain ohmic resistance	Ohm	0	100	*
5	RS	source ohmic resistance	Ohm	0	100	*
6	CGS	zero-bias G-S junction capacitance	F	0	5PF	*
7	CGD	zero-bias G-D junction capacitance	F	0	1PF	*
8	PB	gate junction potential	V	1	0.6	
9	IS	gate junction saturation current	A	1.0E-14	1.0E-14	*
10	KF	flicker noise coefficient	-	0		
11	AF	flicker noise exponent	-	1		
12	FC	coefficient for forward-bias depletion capacitance formula	-	0.5		

7.9. MOSFET Models (both N and P channel)

SPICE provides three MOSFET device models which differ in the formulation of the I-V characteristic. The variable LEVEL specifies the model to be used:

LEVEL=1 -> Shichman-Hodges
LEVEL=2 -> MOS2 (as described in [1])
LEVEL=3 -> MOS3, a semi-empirical model(see [1])

The dc characteristics of the MOSFET are defined by the device parameters VTO, KP, LAMBDA, PHI and GAMMA. These parameters are computed by SPICE if process parameters (NSUB, TOX, ...) are given, but user-specified values always override. VTO is positive (negative) for enhancement mode and negative (positive) for depletion mode N-

channel (P-channel) devices. Charge storage is modeled by three constant capacitors, CGSO, CGDO, and CGBO which represent overlap capacitances, by the nonlinear thin-oxide capacitance which is distributed among the gate, source, drain, and bulk regions, and by the nonlinear depletion-layer capacitances for both substrate junctions divided into bottom and periphery, which vary as the MJ and MJSW power of junction voltage respecively, and are determined by the parameters CBD, CBS, CJ, CJSW, MJ, MJSW and PB. There are two built-in models of the charge storage effects associated with the thin-oxide. The default is the piecewise linear voltage-dependent capacitance model proposed by Meyer. The second choice is the charge-controlled capacitance model of Ward and Dutton [1]. The XQC model parameter acts as a flag and a coefficient at the same time. As the former it causes the program to use Meyer's model whenever larger than 0.5 or not specified, and the charge-controlled model when between 0 and 0.5. In the latter case its value defines the share of the channel charge associated with the drain terminal in the saturation region. The thin-oxide charge storage effects are treated slightly different for the LEVEL=1 model. These voltage- dependent capacitances are included only if TOX is specified in the input description and they are represented using Meyer's formulation.

There is some overlap among the parameters describing the junctions, e.g. the reverse current can be input either as IS (in A) or as JS (in A/m^2). Whereas the first is an absolute value the second is multiplied by AD and AS to give the reverse current of the drain and source junctions respectively. This methodology has been chosen since there is no sense in relating always junction characteristics with AD and AS entered on the device card; the areas can be defaulted. The same idea applies also to the zero-bias junction capacitances CBD and CBS (in F) on one hand, and CJ (in F/m^2) on the other. The parasitic drain and source series resistance can be expressed as either RD and RS (in ohms) or RSH (in ohms/sq.), the latter being multiplied by the number of squares NRD and NRS input on the device card.

	name	parameter	units	default	example
1	LEVEL	model index	-	1	
2	VTO	zero-bias threshold voltage	V	0.0	1.0
3	KP	transconductance parameter	A/V^2	2.0E-5	3.1E-5
4	GAMMA	bulk threshold parameter	$V^{0.5}$	0.0	0.37
5	PHI	surface potential	V	0.6	0.65
6	LAMBDA	channel-length modulation (MOS1 and MOS2 only)	1/V	0.0	0.02
7	RD	drain ohmic resistance	Ohm	0.0	1.0
8	RS	source ohmic resistance	Ohm	0.0	1.0
9	CBD	zero-bias B-D junction capacitance	F	0.0	20FF
10	CBS	zero-bias B-S junction capacitance	F	0.0	20FF
11	IS	bulk junction saturation current	A	1.0E-14	1.0E-15
12	PB	bulk junction potential	V	0.8	0.87
13	CGSO	gate-source overlap capacitance per meter channel width	F/m	0.0	4.0E-11
14	CGDO	gate-drain overlap capacitance per meter channel width	F/m	0.0	4.0E-11
15	CGBO	gate-bulk overlap capacitance per meter channel length	F/m	0.0	2.0E-10
16	RSH	drain and source diffusion sheet resisitance	Ohm/sq.	0.0	10.0

[1] A. Vladimirescu and S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2", ERL Memo No. ERL M80/7,Electronics Research Laboratory, University of California, Berkeley, Oct. 1980.

17	CJ	zero-bias bulk junction bottom cap. per sq-meter of junction area	F/m^2	0.0	2.0E-4
18	MJ	bulk junction bottom grading coef.	-	0.5	0.5
19	CJSW	zero-bias bulk junction sidewall cap. per meter of junction perimeter	F/m	0.0	1.0E-9
20	MJSW	bulk junction sidewall grading coef.	-	0.33	
21	JS	bulk junction saturation current per sq-meter of junction area	A/m^2	1.0E-8	
22	TOX	oxide thickness	meter	1.0E-7	1.0E-7
23	NSUB	substrate doping	$1/cm^3$	0.0	4.0E15
24	NSS	surface state density	$1/cm^2$	0.0	1.0E10
25	NFS	fast surface state density	$1/cm^2$	0.0	1.0E10
26	TPG	type of gate material: +1 opp. to substrate -1 same as substrate 0 Al gate	-	1.0	
27	XJ	metallurgical junction depth	meter	0.0	1U
28	LD	lateral diffusion	meter	0.0	0.8U
29	UO	surface mobility	$cm^2/V-s$	600	700
30	UCRIT	critical field for mobility degradation (MOS2 only)	V/cm	1.0E4	1.0E4
31	UEXP	critical field exponent in mobility degradation (MOS2 only)	-	0.0	0.1
32	UTRA	transverse field coeff (mobility) (deleted for MOS2)	-	0.0	0.3
33	VMAX	maximum drift velocity of carriers	m/s	0.0	5.0E4
34	NEFF	total channel charge (fixed and mobile) coefficient (MOS2 only)	-	1.0	5.0
35	XQC	thin-oxide capacitance model flag and coefficient of channel charge share attributed to drain (0-0.5)	-	1.0	0.4
36	KF	flicker noise coefficient	-	0.0	1.0E-26
37	AF	flicker noise exponent	-	1.0	1.2
38	FC	coefficient for forward-bias depletion capacitance formula - 0.5			
39	DELTA	width effect on threshold voltage (MOS2 and MOS3)	-	0.0	1.0
40	THETA	mobility modulation (MOS3 only)	1/V	0.0	0.1
41	ETA	static feedback (MOS3 only)	-	0.0	1.0
42	KAPPA	saturation field factor (MOS3 only)	-	0.2	0.5

8. SUBCIRCUITS

A subcircuit that consists of SPICE elements can be defined and referenced in a fashion similar to device models. The sub- circuit is defined in the input deck by a grouping of element cards; the program then automatically inserts the group of elements wherever the subcircuit is referenced. There is no limit on the size or complexity of subcircuits, and subcircuits may contain other subcircuits. An example of subcircuit usage is given in Appendix A.

8.1. .SUBCKT Card

General form:

.SUBCKT subnam N1 < N2 N3 ...>

Examples:

```
.SUBCKT OPAMP 1 2 3 4
```

A circuit definition is begun with a .SUBCKT card. SUBNAM is the subcircuit name, and N1, N2, ... are the external nodes, which cannot be zero. The group of element cards which immediately follow the .SUBCKT card define the subcircuit. The last card in a subcircuit definition is the .ENDS card (see below). Control cards may not appear within a vsubcircuit definition; however, subcircuit definitions may contain anything else, including other subcircuit definitions, device models, and subcircuit calls (see below). Note that any device models or subcircuit definitions included as part of a subcircuit definition are strictly local (i.e., such models and definitions are not known outside the subcircuit definition). Also, any element nodes not included on the .SUBCKT card are strictly local, with the exception of 0 (ground) which is always global.

8.2. .ENDS Card

General form:

```
.ENDS <SUBNAM>
```

Examples:

```
.ENDS OPAMP
```

This card must be the last one for any subcircuit definition. The subcircuit name, if included, indicates which subcircuit definition is being terminated; if omitted, all subcircuits being defined are terminated. The name is needed only when nested subcircuit definitions are being made.

8.3. Subcircuit Calls

General form:

```
XYYYYYYY N1 <N2 N3 ...> SUBNAM
```

Examples:

```
X1 2 4 17 3 1 MULTI
```

Subcircuits are used in SPICE by specifying pseudo-elements beginning with the letter X, followed by the circuit nodes to be used in expanding the subcircuit.

9. CONTROL CARDS

9.1. .TEMP Card

General form:

```
.TEMP T1 <T2 <T3 ...> >
```

Examples:

```
.TEMP -55.0 25.0 125.0
```

This card specifies the temperatures at which the circuit is to be simulated. T1, T2, ... Are the different temperatures, in degrees C. Temperatures less than -223.0 deg C are ignored. Model data are specified at TNOM degrees (see the .OPTIONS card for TNOM); if the .TEMP card is omitted, the simulation will also be performed at a temperature equal to TNOM.

9.2. .WIDTH Card

General form:

.WIDTH IN=COLNUM OUT=COLNUM

Examples:

.WIDTH IN=72 OUT=133

COLNUM is the last column read from each line of input; the setting takes effect with the next line read. The default value for COLNUM is 80. The out parameter specifies the output print width. Permissible values for the output print width are 80 and 133.

9.3. .OPTIONS Card

General form:

.OPTIONS OPT1 OPT2 ... (or OPT=OPTVAL ...)

Examples:

.OPTIONS ACCT LIST NODE

This card allows the user to reset program control and user options for specific simulation purposes. Any combination of the following options may be included, in any order. 'x' (below) represents some positive number.

option	effect
ACCT	causes accounting and run time statistics to be printed
LIST	causes the summary listing of the input data to be printed
NOMOD	suppresses the printout of the model parameters.
NOPAGE	suppresses page ejects
NODE	causes the printing of the node table.
OPTS	causes the option values to be printed.
GMIN=x	sets the value of GMIN, the minimum conductance allowed by the program. The default value is 1.0E-12.
RELTOL=x	resets the relative error tolerance of the program. The default value is 0.001 (0.1 percent).
ABSTOL=x	resets the absolute current error tolerance of the program. The default value is 1 picoamp.
VNTOL=x	resets the absolute voltage error tolerance of the program. The default value is 1 microvolt.

TRTOL=x resets the transient error tolerance. The default value is 7.0. This parameter is an estimate of the factor by which SPICE overestimates the actual truncation error. IP "CHGTOL=x" 17 resets the charge tolerance of the program. The default value is 1.0E-14.

PIVTOL=x resets the absolute minimum value for a matrix entry to be accepted as a pivot. The default value is 1.0E-13.

PIVREL=x resets the relative ratio between the largest column entry and an acceptable pivot value. The default value is 1.0E-3. In the numerical pivoting algorithm the allowed minimum pivot value is determined by *EPSREL =AMAX* 1(*PIVREL*·*MAXVAL* ,*PIVTOL*) where MAXVAL is the maximum element in the column where a pivot is sought (partial pivoting).

NUMDGT=x is the number of significant digits printed for output variable values. X must satisfy the relation $0 < x < 8$. The default value is 4. Note: this option is independent of the error tolerance used by SPICE (i.e., if the values of options RELTOL, ABSTOL, etc., are not changed then one may be printing numerical 'noise' for NUMDGT > 4.

TNOM=x resets the nominal temperature. The default value is 27 deg C (300 deg K).

ITL1=x resets the dc iteration limit. The default is 100.

ITL2=x resets the dc transfer curve iteration limit. The default is 50.

ITL3=x resets the lower transient analysis iteration limit. The default value is 4.

ITL4=x resets the transient analysis timepoint iteration limit. The default is 10.

ITL5=x resets the transient analysis total iteration limit. The default is 5000. Set ITL5=0 to omit this test.

ITL6=x resets the dc iteration limit at each step of the source stepping method. The default is 0 which means not to use this method.

CPTIME=x is the maximum cpu-time in seconds allowed for this job.

LIMTIM=x resets the amount of cpu time reserved by SPICE for generating plots should a cpu time-limit cause job termination. The default value is 2 (seconds).

LIMPTS=x resets the total number of points that can be printed or plotted in a dc, ac, or transient analysis. The default value is 201.

LVLCOD=x if x is 2 (two), then machine code for the matrix solution will be generated. Otherwise, no machine code is generated. The default value is 2. Applies only to CDC computers.

LVLTIM=x is 1 (one), the iteration timestep control is used. if x is 2 (two), the truncation-error timestep is used. The default value is 2. If method=Gear and MAXORD> 2 then LVLTIM is set to 2 by SPICE.

METHOD=name sets the numerical integration method used by SPICE. Possible names are Gear or trapezoidal. The default is trapezoidal.

MAXORD=x sets the maximum order for the integration method if Gear's variable-order method is used. X must be between 2 and 6. The default value is 2.

DEFL=x resets the value for MOS channel length; the default is 100.0 micrometer.

DEFW=x resets the value for MOS channel width; the default is 100.0 micrometer.

DEFAD=x resets the value for MOS drain diffusion area; the default is 0.0.

DEFAS=x resets the value for MOS source diffusion area; the default is 0.0.

9.4. .OP Card

General form:

```
.OP
```

The inclusion of this card in an input deck will force SPICE to determine the dc operating point of the circuit with inductors shorted and capacitors opened. Note: a dc analysis is automatically performed prior to a transient analysis to determine the transient initial conditions, and prior to an ac small-signal analysis to determine the linearized, small-signal models for nonlinear devices.

SPICE performs a dc operating point analysis if no other analyses are requested.

9.5. .DC Card

General form:

```
.DC SRCNAM VSTART VSTOP VINCR [SRC2 START2 STOP2 INCR2]
```

Examples:

```
.DC VIN 0.25 5.0 0.25
.DC VDS 0 10 .5 VGS 0 5 1
.DC VCE 0 10 .25 IB 0 10U 1U
```

This card defines the dc transfer curve source and sweep limits. SRCNAM is the name of an independent voltage or current source. VSTART, VSTOP, and VINCR are the starting, final, and incrementing values respectively. The first example will cause the value of the voltage source VIN to be swept from 0.25 Volts to 5.0 Volts in increments of 0.25 Volts. A second source (SRC2) may optionally be specified with associated sweep parameters. In this case, the first source will be swept over its range for each value of the second source. This option can be useful for obtaining semiconductor device output characteristics. See the second example data deck in that section of the guide.

9.6. .NODESET Card

General form:

```
.NODESET V(NODNUM)=VAL V(NODNUM)=VAL ...
```

Examples:

```
.NODESET V(12)=4.5 V(4)=2.23
```

This card helps the program find the dc or initial transient solution by making a preliminary pass with the specified nodes held to the given voltages. The restriction is then released and the iteration continues to the true solution. The .NODESET card may be necessary for convergence on bistable or astable circuits. In general, this card should not be necessary.

9.7. .IC Card

General form:

.IC V(NODNUM)=VAL V(NODNUM)=VAL ...

Examples:

.IC V(11)=5 V(4)=-5 V(2)=2.2

This card is for setting transient initial conditions. It has two different interpretations, depending on whether the UIC parameter is specified on the .TRAN card. Also, one should not confuse this card with the .NODESET card. The .NODESET card is only to help dc convergence, and does not affect final bias solution (except for multi-stable circuits). The two interpretations of this card are as follows:

1. When the UIC parameter is specified on the .TRAN card, then the node voltages specified on the .IC card are used to compute the capacitor, diode, BJT, JFET, and MOSFET initial conditions. This is equivalent to specifying the IC=... parameter on each device card, but is much more convenient. The IC=... parameter can still be specified and will take precedence over the .IC values. Since no dc bias (initial transient) solution is computed before the transient analysis, one should take care to specify all dc source voltages on the .IC card if they are to be used to compute device initial conditions.
2. When the UIC parameter is not specified on the .TRAN card, the dc bias (initial transient) solution will be computed before the transient analysis. In this case, the node voltages specified on the .IC card will be forced to the desired initial values during the bias solution. During transient analysis, the constraint on these node voltages is removed.

9.8. .TF Card

General form:

.TF OUTVAR INSRC

Examples:

.TF V(5,3) VIN
.TF I(VLOAD) VIN

This card defines the small signal output and input for the dc small signal analysis. OUTVAR is the small-signal output variable and INSRC is the small-signal input source. If this card is included, SPICE will compute the dc small signal value of the transfer function (output/input), input resistance, and output resistance. For the first example, SPICE would compute the ratio of V(5,3) to VIN, the small-signal input resistance at VIN, and the small-signal output resistance measured across nodes 5 and 3.

9.9. .SENS Card

General form:

.SENS OV1 < OV2 ... >

Examples:

```
.SENS V(9) V(4,3) V(17) I(VCC)
```

If a .SENS card is included in the input deck, SPICE will determine the dc small-signal sensitivities of each specified output variable with respect to every circuit parameter. Note: for large circuits, large amounts of output can be generated.

9.10. .AC Card

General form:

```
.AC DEC ND FSTART FSTOP
.AC OCT NO FSTART FSTOP
.AC LIN NP FSTART FSTOP
```

Examples:

```
.AC DEC 10 1 10K
.AC DEC 10 1K 100MEG
.AC LIN 100 1 100HZ
```

DEC stands for decade variation, and ND is the number of points per decade. OCT stands for octave variation, and NO is the number of points per octave. LIN stands for linear variation, and NP is the number of points. FSTART is the starting frequency, and FSTOP is the final frequency. If this card is included in the deck, SPICE will perform an ac analysis of the circuit over the specified frequency range. Note that in order for this analysis to be meaningful, at least one independent source must have been specified with an ac value.

9.11. .DISTO Card

General form:

```
.DISTO RLOAD <INTER <SKW2 <REFPWR <SPW2>>>>
```

Examples:

```
.DISTO RL 2 0.95 1.0E-3 0.75
```

This card controls whether SPICE will compute the distortion characteristic of the circuit in a small-signal mode as a part of the ac small-signal sinusoidal steady-state analysis. The analysis is performed assuming that one or two signal frequencies are imposed at the input; let the two frequencies be f1 (the nominal analysis frequency) and f2 (=*SKW* 2·*f* 1). The program hen computes the following distortion measures:

HD2 - the magnitude of the frequency component 2·*f* 1 assuming that f2 is not present.

HD3 - the magnitude of the frequency component 3·*f* 1 assuming that f2 is not present.

SIM2 - the magnitude of the frequency component f1 + f2.

DIM2 - the magnitude of the frequency component f1 - f2.

DIM3 - the magnitude of the frequency component 2·*f* 1 - f2.

RLOAD is the name of the output load resistor into which all distortion power products are to be computed. INTER is the interval at which the summary printout of the contributions of all nonlinear devices to the total distortion is to be printed. If omitted or set to zero, no summary printout will be made. REFPWR is the reference power level used in computing the distortion products; if omitted, a value of 1 mW (that is, dbm) is used. SKW2 is the ratio of f2 to f1. If omitted, a value of 0.9 is used (i.e., f2 = 0.9·*f* 1). SPW2 is the amplitude of f2. If omitted, a value of 1.0 is assumed. he distortion measures HD2, HD3, SIM2, DIM2, and DIM3 may also be be printed and/or plotted (see the description of the .PRINT and .PLOT cards).

9.12. .NOISE Card

General form:

```
.NOISE OUTV INSRC NUMS
```

Examples:

```
.NOISE V(5) VIN 10
```

This card controls the noise analysis of the circuit. The noise analysis is performed in conjunction with the ac analysis (see .AC card). OUTV is an output voltage which defines the summing point. INSRC is the name of the independent voltage or current source which is the noise input reference. NUMS is the summary interval. SPICE will compute the equivalent output noise at the specified output as well as the equivalent input noise at the specified input. In addition, the contributions of every noise generator in the circuit will be printed at every NUMS frequency points (the summary interval). If NUMS is zero, no summary printout will be made.

The output noise and the equivalent input noise may also be printed and/or plotted (see the description of the .PRINT and .PLOT cards).

9.13. .TRAN Card

General form:

```
.TRAN TSTEP TSTOP < TSTART < TMAX> > < UIC>
```

Examples:

```
.TRAN 1NS 100NS
.TRAN 1NS 1000NS 500NS
.TRAN 10NS 1US UIC
```

TSTEP is the printing or plotting increment for line-printer output. For use with the post-processor, TSTEP is the suggested computing increment. TSTOP is the final time, and TSTART is the initial time. If TSTART is omitted, it is assumed to be zero. The transient analysis always begins at time zero. In the interval <zero, TSTART>, the circuit is analyzed (to reach a steady state), but no outputs are stored. In the interval <TSTART, TSTOP>, the circuit is analyzed and outputs are stored. TMAX is the maximum stepsize that SPICE will use (for default, the program chooses either TSTEP or (TSTOP-TSTART)/50.0, whichever is smaller. TMAX is useful when one wishes to guarantee a computing interval which is smaller than the printer increment, TSTEP.

UIC (use initial conditions) is an optional keyword which indicates that the user does not want SPICE to solve for the quiescent operating point before beginning the transient analysis. If this keyword is specified, SPICE uses the values specified using IC=... on the various elements as the initial transient condition and proceeds with the analysis. If the .IC card has been specified, then the node voltages on the .IC card are used to compute the intitial conditions for the devices. Look at the description on the .IC card for its interpretation when UIC is not specified.

9.14. .FOUR Card

General form:

.FOUR FREQ OV1 <OV2 OV3 ...>

Examples:

.FOUR 100K V(5)

This card controls whether SPICE performs a Fourier analysis as a part of the transient analysis. FREQ is the fundamental frequency, and OV1, ..., are the output variables for which the analysis is desired. The Fourier analysis is performed over the interval <TSTOP-period, TSTOP>, where TSTOP is the final time specified for the transient analysis, and period is one period of the fundamental frequency. The dc component and the first nine components are determined. For maximum accuracy, TMAX (see the .TRAN card) should be set to period/100.0 (or less for very high-Q circuits).

9.15. .PRINT Cards

General form:

.PRINT PRTYPE OV1 <OV2 ... OV8>

Examples:

```
.PRINT TRAN V(4) I(VIN)
.PRINT AC VM(4,2) VR(7) VP(8,3)
.PRINT DC V(2) I(VSRC) V(23,17)
.PRINT NOISE INOISE
.PRINT DISTO HD3 SIM2(DB)
```

This card defines the contents of a tabular listing of one to eight output variables. PRTYPE is the type of the analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The form for voltage or current output variables is as follows:

V(N1<,N2>) specifies the voltage difference between nodes N1 and N2. If N2 (and the preceding comma) is omitted, ground (0) is assumed. For the ac analysis, five additional outputs can be accessed by replacing the letter V by:

VR - real part
VI - imaginary part
VM - magnitude
VP - phase
VDB - 20*log*10(magnitude)

I(VXXXXXXX) specifies the current flowing in the independent voltage source named VXXXXXXX. Positive current flows from the positive node, through the source, to the negative node. For the ac analysis, the corresponding replacements for the letter I may be made in the same way as described for voltage outputs.

Output variables for the noise and distortion analyses have a different general form from that of the other analyses, i.e.

OV< (X)>

where OV is any of ONOISE (output noise), INOISE (equivalent input noise), D2, HD3, SIM2, DIM2, or DIM3 (see description of distortion analysis), and X may be any of:

R - real part
I - imaginary part
M - magnitude (default if nothing specified)
P - phase
DB - 20*log*10(magnitude)

thus, SIM2 (or SIM2(M)) describes the magnitude of the SIM2 distortion measure, while HD2(R) describes the real part of the HD2 distortion measure.

There is no limit on the number of .PRINT cards for each type of analysis.

9.16. .PLOT Cards

General form:

.PLOT PLTYPE OV1 < (PLO1,PHI1)> < OV2 < (PLO2,PHI2)> ... OV8>

Examples:

```
.PLOT DC V(4) V(5) V(1)
.PLOT TRAN V(17,5) (2,5) I(VIN) V(17) (1,9)
.PLOT AC VM(5) VM(31,24) VDB(5) VP(5)
.PLOT DISTO HD2 HD3(R) SIM2
.PLOT TRAN V(5,3) V(4) (0,5) V(7) (0,10)
```

This card defines the contents of one plot of from one to eight output variables. PLTYPE is the type of analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The syntax for the OVI is identical to that for the .PRINT card, described above.

The optional plot limits (PLO,PHI) may be specified after any of the output variables. All output variables to the left of a pair of plot limits (PLO,PHI) will be plotted using the same lower and upper plot bounds. If plot limits are not specified, SPICE will automatically determine the minimum and maximum values of all output variables being plotted and scale the plot to fit. More than one scale will be used if the output variable values warrant (i.e., mixing output variables with values which are orders-of-magnitude different still gives readable plots).

The overlap of two or more traces on any plot is indicated by the letter X.

When more than one output variable appears on the same plot, the first variable specified will be printed as well as plotted. If a printout of all variables is desired, then a companion .PRINT card should be included.

There is no limit on the number of .PLOT cards specified for each type of analysis.

10. APPENDIX A: EXAMPLE DATA DECKS

10.1. Circuit 1

The following deck determines the dc operating point and small-signal transfer function of a simple differential pair. In addition, the ac small-signal response is computed over the frequency range 1Hz to 100MEGHz.

```
SIMPLE DIFFERENTIAL PAIR
VCC 7 0 12
VEE 8 0 -12
VIN 1 0 AC 1
RS1 1 2 1K
RS2 6 0 1K
Q1 3 2 4 MOD1
Q2 5 6 4 MOD1
RC1 7 3 10K
RC2 7 5 10K
RE 4 8 10K
.MODEL MOD1 NPN BF=50 VAF=50 IS=1.E-12 RB=100 CJC=.5PF TF=.6NS
.TF V(5) VIN
.AC DEC 10 1 100MEG
.PLOT AC VM(5) VP(5)
.PRINT AC VM(5) VP(5)
.END
```

10.2. Circuit 2

The following deck computes the output characteristics of a MOS- FET device over the range 0-10V for VDS and 0-5V for VGS.

```
MOS OUTPUT CHARACTERISTICS
.OPTIONS NODE NOPAGE
VDS 3 0
VGS 2 0
M1 1 2 0 0 MOD1 L=4U W=6U AD=10P AS=10P
.MODEL MOD1 NMOS VTO=-2 NSUB=1.0E15 UO=550
* VIDS MEASURES ID, WE COULD HAVE USED VDS, BUT ID WOULD BE NEGATIVE
VIDS 3 1
.DC VDS 0 10 .5 VGS 0 5 1
.PRINT DC I(VIDS) V(2)
.PLOT DC I(VIDS)
.END
```

10.3. Circuit 3

The following deck determines the dc transfer curve and the transient pulse response of a simple RTL inverter. The input is a pulse from 0 to 5 Volts with delay, rise, and fall times of 2ns and a pulse width of 30ns. The transient interval is 0 to 100ns, with printing to be done every nanosecond.

```
SIMPLE RTL INVERTER
VCC 4 0 5
VIN 1 0 PULSE 0 5 2NS 2NS 2NS 30NS
RB 1 2 10K
Q1 3 2 0 Q1
RC 3 4 1K
.PLOT DC V(3)
.PLOT TRAN V(3) (0,5)
.PRINT TRAN V(3)
.MODEL Q1 NPN BF 20 RB 100 TF .1NS CJC 2PF
.DC VIN 0 5 0.1
.TRAN 1NS 100NS
.END
```

10.4. Circuit 4

The following deck simulates a four-bit binary adder, using several subcircuits to describe various pieces of the overall circuit.

```
ADDER - 4 BIT ALL-NAND-GATE BINARY ADDER

   *** SUBCIRCUIT DEFINITIONS

.SUBCKT NAND 1 2 3 4
   * NODES: INPUT(2), OUTPUT, VCC
Q1 9 5 1 QMOD
D1CLAMP 0 1 DMOD
Q2 9 5 2 QMOD
D2CLAMP 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDROP 10 3 DMOD
Q5 3 8 0 QMOD
.ENDS NAND

.SUBCKT ONEBIT 1 2 3 4 5 6
   * NODES: INPUT(2), CARRY-IN, OUTPUT, CARRY-OUT, VCC
X1 1 2 7 6 NAND
X2 1 7 8 6 NAND
X3 2 7 9 6 NAND
X4 8 9 10 6 NAND
X5 3 10 11 6 NAND
X6 3 11 12 6 NAND
X7 10 11 13 6 NAND
X8 12 13 4 6 NAND
X9 11 7 5 6 NAND
.ENDS ONEBIT
```

```
.SUBCKT TWOBIT 1 2 3 4 5 6 7 8 9
   *  NODES:  INPUT - BIT0(2) / BIT1(2), OUTPUT - BIT0 / BIT1,
   *          CARRY-IN, CARRY-OUT, VCC
X1 1 2 7 5 10 9 ONEBIT
X2 3 4 10 6 8 9 ONEBIT
.ENDS TWOBIT
.SUBCKT FOURBIT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
 * NODES: INPUT - BIT0(2) / BIT1(2) / BIT2(2) / BIT3(2),
 *       OUTPUT - BIT0/BIT1/BIT2/BIT3,CARRY-IN,CARRY-OUT,VCC
X1 1 2 3 4 9 10 13 16 15 TWOBIT
X2 5 6 7 8 11 12 16 14 15 TWOBIT
.ENDS FOURBIT

  *** DEFINE NOMINAL CIRCUIT

.MODEL DMOD D
.MODEL QMOD NPN(BF=75 RB=100 CJE=1PF CJC=3PF)
VCC 99 0 DC 5V
VIN1A 1 0 PULSE(0 3 0 10NS 10NS   10NS   50NS)
VIN1B 2 0 PULSE(0 3 0 10NS 10NS   20NS  100NS)
VIN2A 3 0 PULSE(0 3 0 10NS 10NS   40NS  200NS)
VIN2B 4 0 PULSE(0 3 0 10NS 10NS   80NS  400NS)
VIN3A 5 0 PULSE(0 3 0 10NS 10NS  160NS  800NS)
VIN3B 6 0 PULSE(0 3 0 10NS 10NS  320NS 1600NS)
VIN4A 7 0 PULSE(0 3 0 10NS 10NS  640NS 3200NS)
VIN4B 8 0 FULSE(0 3 0 10NS 10NS 1280NS 6400NS)
X1 1 2 3 4 5 6 7 8 9 10 11 12 0 13 99 FOURBIT
RBIT0 9 0 1K
RBIT1 10 0 1K
RBIT2 11 0 1K
RBIT3 12 0 1K
RCOUT 13 0 1K
.PLOT TRAN V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)
.PLOT TRAN V(9) V(10) V(11) V(12) V(13)
.PRINT TRAN V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)
.PRINT TRAN V(9) V(10) V(11) V(12) V(13)

  *** (FOR THOSE WITH MONEY (AND MEMORY) TO BURN)
.TRAN 1NS 6400NS

.OPTIONS ACCT LIST NODE LIMPTS=6401
.END
```

10.5. Circuit 5

The following deck simulates a transmission-line inverter. Two transmission-line elements are required since two propagation modes are excited. In the case of a coaxial line, the first line (T1) models the inner conductor with respect to the shield, and the second line (T2) models the shield with respect to the out- side world.

```
TRANSMISSION-LINE INVERTER
V1 1 0 PULSE(0 1 0 0.1N)
R1 1 2 50
X1 2 0 0 4 TLINE
R2 4 0 50
.SUBCKT TLINE 1 2 3 4
T1 1 2 3 4 Z0=50 TD=1.5NS
T2 2 0 4 0 Z0=100 TD=1NS
.ENDS TLINE
.TRAN 0.1NS 20NS
.PLOT TRAN V(2) V(4)
.END
```

11. APPENDIX B: NONLINEAR DEPENDENT SOURCES

SPICE allows circuits to contain dependent sources charac terized by any of the four equations

i=f(v) v=f(v) i=f(i) v=f(i)

where the functions must be polynomials, and the arguments may be multidimensional. The polynomial functions are specified by a set of coefficients p0, p1, ..., pn. Both the number of dimensions and the number of coefficients are arbitrary. The meaning of the coefficients depends upon the dimension of the polynomial, as shown in the following examples:

Suppose that the function is one-dimensional (that is, a function of one argument). Then the function value fv is determined by the following expression in a (the function argument):

$$fv = p0 + p1 \cdot a + p2 \cdot a^2 + p3 \cdot a^3 + p4 \cdot a^4 + p5 \cdot a^5 + \cdots$$

Suppose now that the function is two-dimensional, with arguments a and b. Then the function value fv is determined by the following expression:

$$fv = p0 + p1 \cdot a + p2 \cdot b + p3 \cdot a^2 + p4 \cdot a \cdot b + p5 \cdot b^2 + p6 \cdot a^3 +$$
$$p7 \cdot a^2 b + p8 \cdot a \cdot b^2 + p9 \cdot b^3 + \cdots$$

Consider now the case of a three-dimensional polynomial function with arguments a, b, and c. Then the function value fv is determined by the following expression:

$$fv = p0 + p1 \cdot a + p2 \cdot b + p3 \cdot c + p4 \cdot a^2 + p5 \cdot a \cdot b +$$
$$p6 \cdot a \cdot c + p7 \cdot b^2 + p8 \cdot b \cdot c + p9 \cdot c^2 + p10 \cdot a^3 +$$
$$p11 \cdot a^2 \cdot b + p12 \cdot a^2 \cdot c + p13 \cdot a \cdot b^2 + p14 \cdot a \cdot b \cdot c +$$
$$p15 \cdot a \cdot c^2 + p16 \cdot b^3 + p17 \cdot b^2 \cdot c +$$
$$p19 \cdot c^3 + p20 \cdot a^4 + \cdots$$

Note: if the polynomial is one-dimensional and exactly one coefficient is specified, then SPICE assumes it to be p1 (and p0 = 0.0), in order to facilitate the input of linear controlled sources.

For all four of the dependent sources described below, the initial condition parameter is described as optional. If not specified, SPICE assumes 0 the initial condition for dependent sources is an initial initial condition to obtain the dc operating point of the circuit. After convergence has been obtained, the program continues iterating to obtain the exact value for the controlling variable. Hence, to reduce the computational effort for the dc operating point (or if the polynomial specifies a strong nonlinearity), a value fairly close to the actual controlling variable should be specified for the initial condition.

11.1. Voltage-Controlled Current Sources

General form:

GXXXXXXX N+ N- <POLY(ND)> NC1+ NC1- ... P0 <P1 ...> <IC=...>

Examples:

```
G1 1 0 5 3 0 0.1M
GR 17 3 17 3 0 1M 1.5M IC=2V
GMLT 23 17 POLY(2) 3 5 1 2 0 1M 17M 3.5U IC=2.5, 1.3
```

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. NC1+, NC1-, ... Are the positive and negative controlling nodes, respectively. One pair of nodes must be specified for each dimension. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling voltage(s). If not specified, 0.0 is assumed. The polynomial specifies the source current as a function of the controlling voltage(s). The second example above describes a current source with value

$$I = 10^{-3} \cdot V(27,3) + 1.5 \times 10^{-3} \cdot V(17,3)^2$$

note that since the source nodes are the same as the controlling nodes, this source actually models a nonlinear resistor.

11.2. Voltage-Controlled Voltage Sources

General form:

EXXXXXXX N+ N- <POLY(ND)> NC1+ NC1- ... P0 <P1 ...> <IC=...>

Examples:

```
E1 3 4 21 17 10.5 2.1 1.75
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1 IC=1.5,2.0,17.35
```

N+ and N- are the positive and negative nodes, respectively. POLY(ND) only has to be specified if the source is multi- dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. NC1+, NC1-, ... are the positive and negative controlling nodes, respectively. One pair of nodes must be specified for each dimension. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling voltage(s). If not specified, 0.0 is assumed. The polynomial specifies the source voltage as a function of the controlling voltage(s). The second example above describes a voltage source with value

$$V = V(13,0) + V(15,0) + V(17,0)$$

(in other words, an ideal voltage summer).

11.3. Current-Controlled Current Sources

General form:

FXXXXXXX N+ N- <POLY(ND)> VN1 <VN2 ...> P0 <P1 ...> <IC=...>

Examples:

```
F1 12 10 VCC 1MA 1.3M
FXFER 13 20 VSENS 0 1
```

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. VN1, VN2, ... are the names of voltage sources through which the controlling current flows; one name must be specified for each dimension. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of each voltage source. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling current(s) (in Amps). If not specified, 0.0 is assumed. The polynomial specifies the source current as a function of the controlling current(s). The first example above describes a current source with value

$$I = 10^{-3} + 1.3 \times 10^{-3} \cdot (VCC)$$

11.4. Current-Controlled Voltage Sources

General form:

```
HXXXXXXX N+ N- <POLY(ND)> VN1 <VN2 ...> P0 <P1 ...> <IC=...>
```

Examples:

```
HXY 13 20 POLY(2) VIN1 VIN2 0 0 0 0 1 IC=0.5 1.3
HR 4 17 VX 0 0 1
```

N+ and N- are the positive and negative nodes, respectively. POLY(ND) only has to be specified if the source is multi- dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. VN1, VN2, ... are the names of voltage sources through which the controlling current flows; one name must be specified for each dimension. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of each voltage source. P0, P1, P2, ..., Pn are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling current(s) (in Amps). If not specified, 0.0 is assumed. The polynomial specifies the source voltage as a function of the controlling current(s). The first example above describes a voltage source with value

$$V = I(VIN1) \cdot I(VIN2)$$

12. APPENDIX C: BIPOLAR MODEL EQUATIONS

Acknowledgment: This section has been contributed by Bill Bider- mann at HP labs.

(G_{min} *terms omitted*)

12.1 D.C. MODEL

$$IC = \frac{IS}{QB}\left(e^{\frac{qV_{BE}}{(NF)kT}} - e^{\frac{qV_{BC}}{(NR)kT}}\right) - \frac{IS}{BR}\left(e^{\frac{qV_{BC}}{(NR)kT}} - 1\right) - ISC\left(e^{\frac{qV_{BC}}{(NC)kT}} - 1\right)$$

$$IB = \frac{IS}{BF}\left(e^{\frac{qV_{BE}}{(NF)kT}} - 1\right) + \frac{IS}{BR}\left(e^{\frac{qV_{BC}}{(NR)kT}} - 1\right) + ISE\left(e^{\frac{qV_{BE}}{(NE)kT}} - 1\right) + ISC\left(e^{\frac{qV_{BC}}{(NC)kT}} - 1\right)$$

NOTE: The last two terms in the expression of the base current IB represent the components due to recombination in the BE and BC space charge regions at low injection.

If IRB not specified

$$RB\bar{B} = RBM + \frac{RB - RBM}{QB}$$

If IRB specified

$$RB\bar{B} = 3(RB - RBM)\left[\frac{\tan(z) - z}{z\tan^2(z)} + RBM\right]$$

Where:

$$z = \frac{-1 + \sqrt{(144IB/(\pi^2 IRB) + 1)}}{24/\pi^2\sqrt{IB/IRB}}$$

$$QB = \frac{Q1}{2}(1 + \sqrt{(1 + 4Q2)})$$

$$Q1 = \frac{1}{1 - \frac{V\bar{B}\bar{C}}{VAF} - \frac{V\bar{B}\bar{E}}{VAR}}$$

$$Q2 = \frac{IS}{IKF}(e^{\frac{qV\bar{B}\bar{E}}{(NF)kT}} - 1) + \frac{IS}{IKR}(e^{\frac{qV\bar{B}\bar{C}}{(NR)kT}} - 1)$$

NOTE: IRB is the current where the base resistance falls halfway to its minimum value. VAF and VAR are forward and reverse Early voltages respectively. IKF and IKR determine the high current beta rolloff with IC. ISE, ISC, NE and NC determine the low current beta rolloff with IC.

12.2 A.C. MODEL

$$CBE = \frac{\partial}{\partial V\bar{B}\bar{E}}\left[(TFF\frac{IS}{QB}(e^{\frac{qV\bar{B}\bar{E}}{(NF)kT}} - 1) + CJE\left(1 - \frac{V\bar{B}\bar{E}}{VJE}\right)^{-MJ}\right]$$

Where:

$$TFF = TF\left(1 + XTF \cdot \left(\frac{IF}{IF + ITF}\right)^2\right) \cdot e^{\frac{V\bar{B}\bar{C}}{1.44VTF}}$$

$$IF = IS(e^{\frac{qV\bar{B}\bar{E}}{(NF)kT}} - 1)$$

$$CBI = CBC \cdot (1 - XCJC)$$

$$CB2 = CBC \cdot XCJC$$

$$CBC = TR\left(\frac{qIS}{(NR)kT}e^{\frac{qV\bar{B}\bar{C}}{kT}}\right) + CJC\left(1 - \frac{V\bar{B}\bar{C}}{VJC}\right)^{-MJC}$$

$$CSS = CJS\left(1 - \frac{V\bar{C}\bar{S}}{VJS}\right)^{-MJS}$$

NOTE: all junction capacitances of the form $CO \cdot (1 - \frac{V}{\phi})^{-M}$ revert to the form

$$\frac{CO}{(1 - FC)^M}\left(1 + \frac{M(V - FC \cdot \phi)}{\phi(1 - FC)}\right)$$

when $V > FC \cdot phi$ (For CSS assumes FC = 0)

12.3 NOISE MODEL

Thermal Noise :

$$IRB\bar{B}^2=\frac{4kT}{R}B\bar{B}\Delta f$$

$$IRC^2=\frac{4kT}{RC}\Delta f$$

$$IRE^2=\frac{4kT}{2RED}\Delta f+\frac{KF\cdot IB\cdot AF}{f}\Delta f$$

Note: The first two terms are shot noise and the last term is flicker noise.

$$ICN^2=2qIC\,\Delta f$$

Note: This is shot noise.

12.4 TEMPERATURE EFFECTS

All junctions have dependences identical to the diode model but all N factors are considered equal 1.

BF and BR go as $(\frac{T}{TNOM})^{XTB}$ when NF=1. This is done through appropriate changes in BF , BR and ISE, ISC according to the following equations respectively:

$$B\bar{F}=BF\cdot(\frac{T}{TNOM})^{XTB}$$

$$B\bar{R}=BR\cdot(\frac{T}{TNOM})^{XTB}$$

$$IS\bar{E}=ISE\cdot(\frac{T}{TNOM})^{(XTI-XTB)}e^{\frac{qEG}{Nk}\frac{T-TNOM}{T\cdot TNOM}}$$

$$IS\bar{C}=ISC\cdot(\frac{T}{TNOM})^{(XTI-XTB)}e^{\frac{qEG}{Nk}\frac{T-TNOM}{T\cdot TNOM}}$$

12.5 EXCESS PHASE

This is a delay (linear phase) in the gm generator in AC analysis. It is also used in transient analysis using a Bessel polynomial approximation. Excess phase, PTF, is specified as the number of extra degrees of phase at the frequency

$$f=\frac{1}{2\pi\cdot TF}Hertz$$

12. APPENDIX D: ALTER STATEMENT AND THE SOURCE-STEPPING METHOD

The ALTER statement allows SPICE to run with altered circuit parameters.

General form:

```
.ALTER
ELEMENT CARDS (DEVICE CARDS, MODEL CARDS)
.ALTER (or .END CARD)
```

Examples:

```
R1 1 0 5K
VCC 3 0 10
```

```
M1 3 2 0 MOD1 L=5U W=2U
.MODEL MOD1 NMOS(VTO=1.0 KP=2.0E-5 PHI=0.6 NSUB=2.0E15 TOX=0.1U)
.ALTER
R1 1 0 3.5K
VCC 3 0 12
M1 3 2 0 MOD1 L=10U W=2U
.MODEL MOD1 NMOS(VTO=1.2 KP=2.0E-5 PHI=0.6 NSUB=5.0E15 TOX=1.5U)
.ALTER
M1 3 2 0 MOD1 L=10U W=4U
.END
```

This card introduces the element(s), device(s) and model(s) whose parameters are changed during the execution of the input deck. The vanalyses specified in the deck will start over again with the changed parameters. The .ALTER card with the cards defining the new parameters should be placed just before the .END card. The syntax for the element (device, model) cards is identical to that of the cards with the original parameters.

There is no limit on the number of .ALTER cards and the circuit will be re-analyzed as many times as the number of .ALTER cards. Subsequent ALTER operations employ parameters of the previous change. No topological change of the circuit is allowed.

The source-stepping method can enhance DC convergence. But it is slower than direct use of the Newton-Raphson method. Therefore it is best used as an alternative to achieve convergence of DC operating point when the circuit fails to converge by using the Newton-Raphson method. The source-stepping method is used by SPICE when the variable ITL6 in the .OPTIONS card is set to the iteration limit at each step of the source(s).